

# DISEÑO Y CONSTRUCCIÓN DE ALGORITMOS

Alfonso Mancilla Herrera
Roberto Ebratt Gómez
José Capacho Portilla






# Diseño y construcción de algoritmos

# Diseño y construcción de algoritmos

Alfonso Mancilla Herrera
Roberto Ebratt Gómez
José Capacho Portilla


Barranquilla
(Colombia), 2016






Mancilla Herrera, Alfonso.
Diseño y construcción de algoritmos / Alfonso Mancilla Herrera, Roberto Ebratt Gómez. José Capacho Portilla. — Barranquilla, Col. : Editorial Universidad del Norte, reimp., 2016.

476 p. ; 24 cm.
Incluye resumen e índice alfabético
ISBN 978-958-741-557-5 (impreso)
ISBN 978-958-741-497-4 (PDF)

I. Programación de computadores. II. Estructura de datos (Computadores). 1. Ebratt Gómez, Roberto. 2. Capacho Portilla, José. 4.Tít.
(005.115 M269 23 ed.) (CO-BrUNB)




www.uninorte.edu.co
Km 5, vía a Puerto Colombia
A.A. 1569, Barranquilla (Colombia)




www.edicionesdelau.com
Transversal 42 n.° 4B-83, B. Primavera
Bogotá (Colombia)



Primera edición, enero de 2015
Primera reimpresión, noviembre de 2015
Segunda reimpresión, mayo de 2016

*Coordinación editorial*
Zoila Sotomayor O.

*Diagramación*
Alfonso Mancilla Herrera

*Diseño de portada*
Silvana Pacheco

*Corrección de textos*
Henry Stein

*Procesos técnicos*
Munir Kharfan de los Reyes

Impreso y hecho en Colombia
Nombre de la imprenta (Bogotá)
*Printed and made in Colombia*

*A*
*Dios,*
*mi esposa e hijos,*
*mis padres y hermanos.*

Alfonso Mancilla

*A Dios,*
*mi padres y*
*Jessica.*

Roberto Ebratt

*A Dios,*
*mi madre Evila,*
*mi esposa Juanita.*

José Capacho

# Índice general

# Índice de figuras

# Lista de ejemplos

# Preliminares

## Siglas y abreviaturas usadas

| | | |
|---|---|---|
| ABET | : | Accreditation Board for Engineering and Technology. |
| ASCII | : | American Standard Code for Information Interchange. |
| ACM/IEEE-CS | : | Association for Computing Machinery / Institute of Electrical and Electronics Engineers-Computer Society. |
| BIT | : | Binary digit. |
| CPU | : | Central Processing Unit. |
| IBM | : | International Business Machines. |
| RAM | : | Random Access Memory. |
| ROM | : | Read Only Memory. |
| SDF | : | Software Development Fundamentals. |

# Introducción

Los contextos políticos, sociales, económicos y ambientales del mundo actual tienen en la sociedad del conocimiento un recurso fundamental de decisión: la información. La sociedad digital se fundamenta para sus decisiones en grandes bancos de datos de información, los cuales se conectan y comparten información a través de la macrored Internet. Los bancos de información para su creación y desarrollo se fundamentan tanto en la Ingeniería de Software como en la Ingeniería de Hardware.

La Ingeniería de Software se centra en la creación de estructuras lógicas de control que gobiernan y permiten el funcionamiento de los sistemas informatizados; por su parte, la Ingeniería de Hardware tiene como finalidad la construcción de máquinas electrónicas de procesamiento de datos que cumplen las funciones de conectividad, comunicación y transformación de los datos en informaciones con base en el software, como es el caso de una red de computadores.

El software, siendo la lógica de control de los sistema informáticos, se basa tanto en la lógica humana como en la lógica algorítmica para su creación; lógicas que se complementan con la lógica de programación al pasar el algoritmo a un lenguaje de programación para correrlo en un computador digital, en la óptica de transformar los datos en información. La interrelación software-hardware en su operación genera un sistema computacional basado en máquinas que procesan datos electrónicamente. Sistema computacional que se enmarca a nivel distribuido en un sistema mayor que funciona a nivel distribuido que es la red (Internet); red que al estar al servicio de una comunidad hace que la actual sociedad esté compuesta por grupos sociales cuyas acciones de conectividad, comunicación y procesamiento se caracterizan por la alta dinámica de generación de informaciones a través de la red; información que está al servicio de grupos sociales. Lo anterior justifica el sentido de sociedad digital. Dada la cobertura de la red en su alcance global al realizar sus procesos, la sociedad actual, además de ser digital, es también global, porque la red, al ser distribuida, permite la conectividad y el paso de informaciones entre puntos geográficos remotos o distantes ubicados no solo en el planeta Tierra sino fuera de él.

Debido a que la sociedad actual es una sociedad digital y global, también está inserta en un paradigma socio-técnico, en su doble sentido, justificado por el hecho de que son las redes sociales las artífices o generadoras de informaciones y es la ciencia (*Computer Science* o Ciencia de la Computación) y la técnica las que han posibilitado el notorio avance de la Informática.

Los datos procesados a través del software, al ser convertidos en informaciones, permiten la toma de decisiones. Decisiones que al ser influenciadas por nuevas informaciones dan como resultado conocimiento humano; y allí estriba la justificación de la sociedad del conocimiento.

Luego, la sociedad digital y global, tendiente a ser una sociedad del conocimiento, ha sido influenciada en su desarrollo por la lógica algorítmica, la cual se traduce en lógica de programación cuando el algoritmo se codifica en un lenguaje entendible

para el computador. Lógicas que se basan en el razonamiento humano. Por lo tanto, se hace necesario para el actual y el futuro profesional en cualquier disciplina, y más específicamente para los profesionales de la Ingeniería, el conocimiento de los conceptos teórico-prácticos de la lógica algorítmica y, consecuentemente, la lógica de programación.

Con base en lo anterior, y con relación a este texto, la lógica de programación se desarrolla de acuerdo con la siguiente estructura:

Primero, se construye el marco conceptual de la lógica de programación. Segundo, se presentan las estructuras básicas lógicas para la construcción de algoritmos. Tercero, se desarrollan los conceptos de las estructuras de datos básicas, para ser manejadas por sus lógicas de control algorítmicas asociadas, luego se continúa con el desarrollo del tópico de la lógica de programación modular, al término de cuyo contenido se presenta una sección de problemas desafiantes, la sección de conclusiones y la bibliografía utilizada en la redacción de esta obra.

# Justificación

Los sistemas de información al servicio de la sociedad del conocimiento se basan tanto en la Ciencia de la Computación (*Computer Science*) como en la Ingeniería de Software (*Software Engineering*). La resolución de problemas del mundo real de cualquier disciplina y a cualquier nivel en las escalas económicas nacional e internacional han utilizado los sistemas generadores de información en la captura, almacenamiento y transformación de datos almacenados en los sistemas de cómputos con el propósito de dar informaciones útiles al usuario. Es así como el Software y la Ciencia de la Computación han estado y estarán presentes en la solución de problemas relacionados con la exploración espacial, el control del clima, la conservación del planeta, la salud del ser humano, la educación de los ciudadanos, la economía de un sistema de país o unión y los sistemas de noticias cuyas unidades digitales dan a conocer al instante el estado actual político, social y económico del universo-mundo.

Luego, cualquier sistema informático basado en software que opere sobre un conjunto de máquinas de procesamiento electrónico de dados (computadores) depende de los algoritmos para su funcionamiento. Sin hacer una enumeración exhaustiva, las áreas de conocimiento relacionadas con la Ciencia de la Computación, tales como: Computación Gráfica, Interacción Hombre-Máquina, Administración de la Información, Sistemas Inteligentes, Sistemas Operacionales, Lenguajes de Programación, o Redes de computación, no tienen un funcionamiento en el mundo real sin su relación con los algoritmos.

En tal sentido, los sistemas de visualización en espacios bidimensionales y tridimensionales necesitan de los algoritmos que transformen las funciones matemáticas, representaciones de las áreas y volúmenes en imágenes gráficas presentadas a través del computador. Los sistemas de comunicación hombre-máquina en su teoría de interfaces comunicativas requieren de los algoritmos para manejar el proceso de comunicación y entendimiento entre el hombre, quien enviando datos al computador espera de la máquina una respuesta útil transformada en información. En el marco de la administración de la información, los sistemas de bases de datos requieren de algoritmos capaces de manejar las acciones organización, mantenimiento y consulta de la base de datos utilizando un gestor de bases de datos. Los sistemas inteligentes en sus procesos de razonamiento dependen de los algoritmos para la solución de casos de reconocimiento de voz, visión por computador o actuadores a través de máquinas robots. Los sistemas operacionales utilizan algoritmos en sus procesos de concurrencia, administración de memoria, organización de máquinas virtuales o administración de dispositivos de entrada y salida. Los lenguajes de programación no tendrían ningún sentido sin los algoritmos; porque es precisamente a partir de los algoritmos que un lenguaje de programación es funcional; y más aun, el mismo proceso de traducción de un lenguaje de programación de alto nivel (Java, C, C++, ...) a lenguaje de máquina (el que entiende el computador (unos y ceros)) depende de los algoritmos que componen las piezas de software del traductor o compilador. Los procesos de conectividad y comunicación presentes en las redes computaciones no

son posibles sin los algoritmos de localización y enrutamiento al operar un sistema distribuido funcionado con una determinada topología de red.

Por su parte, la Ingeniería de Software en sus dominios específicos relacionados con la Ciencias de la Computación o con las aplicaciones de negocios necesariamente requiere algoritmos para su funcionamiento. Teniendo en cuenta que la Ingeniería de Software es

> ... la disciplina concerniente con la aplicación de teoría, conocimiento y práctica a la efectiva y eficiente construcción confiable de sistemas de software que satisfagan los requerimientos de clientes y usuarios. (Computer Science Curricula, 2013, p. 143),

Las acciones en los campos de Ingeniería de Requisitos, Arquitecturas de Software, Gerencia de un Proyecto de Software y Sistemas relacionados con el aseguramiento de la Calidad del Software se basan para el desarrollo de sus procesos en los algoritmos. Lo anterior justificado en términos de los aportes de la Ingeniería de Software en su relación con aplicaciones de negocios en: Ingeniería Industrial, al utilizar sistemas en tiempo real para el control de una línea de producción automatizada, porque todos los procesos de la línea de fabricación en su interrelación hardware-sofware están soportados por algoritmos; Ingeniería Electrónica, al utilizar procesos de comunicación y conectividad en computación móvil, porque los procesos de emisor-receptor de contacto y transferencias de informaciones entre dispositivos móviles se basan en algoritmos; Ingeniería Mecánica, al utilizar sistemas gráficos para el diseño de elementos mecánicos, porque son necesarios algoritmos para los sistemas de diseño 2D o 3D; Ingeniería Civil, por la utilización de sistemas de información que apoyen el cálculo de elementos finitos en una estructura de obra civil, porque dichos sistemas no pueden funcionar sin algoritmos; Ingeniería Eléctrica, por el empleo de sistemas distribuidos para el control, generación y distribución de la energía eléctrica; sistemas que dependen necesariamente de los algoritmos para el logro de la integración funcional del sistema energético en sus componentes eléctrica-mecánica-electrónica-informática. Complementado lo anterior con la aplicación de la Ingeniería de Software a otras áreas del conocimiento diferentes a la Ingeniería, como es el caso de: Educación, por la utilización de los sistemas basados en la web, cuyo funcionamiento para cumplir con procesos de teleformación dependen necesariamente de los algoritmos; Medicina, en el uso de sistemas altamente integrados empleados en telemedicina, cuyos procesos de sensórica computacional no serían posibles sin el empleo de algoritmos; Administración, en la utilización de la teoría de juegos computacionales, para asegurar a través de simulaciones inversiones finacieras bajo condiciones de riesgo; juegos y simulaciones que necesariamente funcionan con base en algoritmos probabilísticos, como son los algoritmos de Monte Carlo, Las Vegas y los Probabilístas Numéricos (Brassard & Bratley, 1997, pp. 325-409).

Habiendo justificado la importancia de los algoritmos con relación tanto a la Ciencia de la Computación como a la Ingeniería de Software, es relevante para la resolución de problemas del mundo real factibles de ser tratados computacionalmente el aprendizaje del diseño de algoritmos. Lo anterior, como lo expresa Luis Joyanes

Aguilar en el prólogo a la edición en español del libro *Fundamentos de Algoritmia*, se reafirma por el hecho de que

> ... el estudio del concepto, diseño y construcción de algoritmos será una necesidad ineludible en numerosos aspectos de la vida, y esencialmente en el mundo científico y de la ingeniería. Así lo han entendido todos los organismos nacionales e internacionales relacionados con el mundo de la educación, que han aconsejado siempre la inclusión de asignaturas específicas de Algoritmia, y en particular de análisis y diseño de algoritmos, para los diferentes curricula... (Brassard & Bratley, 1997, p. xxi)

Reconociendo la importancia tanto de los algoritmos como de su aprendizaje, el libro que el lector tiene en sus manos enfatiza en el diseño y construcción de algoritmos en una lógica de programación que sea factible de ser codificada en un lenguaje de programación en un computador.

Con base en la interrelación mencionada, este libro profundiza en la presentación de cada cada una de las estructuras lógicas que sirven para el diseño de algoritmos. Las estructuras lógicas son presentadas en una notación formal, pero comprensible a usuarios sin ningún conocimiento de matemáticas, algoritmos ni programación; el desarrollo de la notación mencionada en el proceso de diseño de los algoritmos se basa en la metodología de resolución de problemas; para lo cual los problemas son ejemplificados, presentando en primer lugar la identidad del problema; en segundo lugar, el análisis del problema utilizando notaciones gráfico - matemáticas para dar un mayor entendimiento al lector, y siendo el análisis para la resolución del problema un valor agregado con relación a otros textos de la misma especie; finalmente, se presenta la propuesta de diseño de solución al problema previamente identificado y analizado en notación algorítmica, lo cual se complementa con comentarios que explican los pasos seguidos por el algoritmo al resolver el problema. Los autores consideran que el diseño que se presenta para cada problema está en términos de propuesta de solución; porque la solución de un problema del mundo real a través de algoritmos no es única, aceptando los diferentes enfoques de solución algorítmica que puede tener un problema.

El enfoque empleado para la resolución de problemas (*problem solving*) se da a nivel didáctico, en un orden creciente de dificultad; se hace la salvedad de que este texto no es un manual de programación, porque ni siquiera se codifica un solo algoritmo en ningún lenguaje de programación; tampoco, la teoría relacionada con las estructuras lógicas y sus aplicaciones constituyen un conjunto de pasos para ser aplicado como un "recetario de cocina", porque este texto privilegia el análisis del problema para ser expresado con base en la lógica humana a través de un conjunto de estructuras de control representativas de la lógica algorítmica.

# Síntesis conceptual de cada capítulo

Los sistemas informáticos de apoyo a las decisiones en las organizaciones requieren de la construcción de bancos de datos estructurados sobre un hardware y funcionando mediante un software que permite la conversión de los datos almacenados en un computador en información para tomar acciones de gestión - decisión útiles al usuario.

En el marco de los sistemas de información al servicio de la sociedad, este libro está estructurado en seis capítulos, los cuales integran los conceptos esenciales relacionados con la base primigenia de la fabricación de software como es la lógica humana y la algoritmia. Lo anterior se complementa con los fundamentos esenciales de la arquitectura de un computador como base física para el almacenamiento tanto de los datos como de los programas escritos en un lenguaje de programación.

El primer capítulo proporciona al lector los conceptos básicos de lógica y algoritmos. La lógica humana se desarrolla a partir de una base biológica (el cerebro) para llegar a un sistema de razonamiento formal que interactúa con la Algoritmia. Los conceptos de algoritmos se escriben a partir de su desarrollo histórico, para después pasar a definir el concepto de sistema y posteriormente desarrollar el concepto de sistema en el marco de la Teoría General de Sistemas (TGS), centrando su significado como unidad sistémica generadora de información. Teniendo como fundamento los conceptos de sistema, lógica y algoritmo, este texto desarrolla el concepto de algoritmo en el marco de la lógica humana racional, lo que conlleva al concepto de la lógica algorítmica.

Una vez dado el fundamento de interrelación sistema-lógica-algoritmo en su integración, continúa el capítulo en mención el desarrollo de las máquinas de procesamiento electrónico de datos concretando las bases de la organización computacional en sus tópicos relacionado con: el sistema computacional, el concepto de computador y su arquitectura, el almacenamiento de datos en un computador y los sistemas de numeración necesarios para entender el almacenamiento de los datos en el computador. Finalmente, teniendo los fundamentos integrados de sistema-lógica-algoritmo-computador, se desarrolla la lógica de programación en Ingeniería.

El tercer capítulo proporciona al lector las herramientas básicas necesarias para la construcción de algoritmos. Las herramientas están fundamentadas en estructuras lógicas básicas, en función de las cuales el lector puede desarrollar sus propios algoritmos, aplicados a cualquier campo del conocimiento humano, dado que son conceptos de lógicas esenciales de apoyo a la algoritmia. El desarrollo de las estructuras lógicas se inicia con las relacionadas a la entrada, la salida y la asignación como punto inicial para la presentación del concepto de primitivas básicas para la construcción de algoritmos; se aclara que el término “primitivas” no se debe asociar con estructuras antiguas, sino que el mensaje en este caso es que son como los adobes de un edificio o la célula del cuerpo humano, que para el caso de la algorítmia son las estructuras de lógicas primigenias en las cuales se basa el diseño de algoritmos.

Con base en lo anterior, el capítulo continúa su desarrollo con las estructuras lógicas condicionales simples, compuestas, *Dependiendo_De*, repetiviva *Para*, y finalmente termina con las lógicas de control *Mientras_Que* y *Haga_Hasta*. Cada una de las lógicas desarrolla un conjunto de problemas, respecto de los cuales es importante resaltar que en su resolución algorítmica se incluye para cada uno de ellos la definición del problema; el análisis del problema, por cuanto es precisamente el análisis del problema lo que fundamenta el diseño del algoritmo a través de la estructura lógica; y finalmente se presenta el diseño concreto de cada algoritmo. Diseño que se complementa con un conjunto de comentarios asociados a cada parte de la lógica de control del algoritmo.

El capítulo cuatro presenta las estructuras de datos de vectores unidimensionales y arreglos bidimenionales o matrices con sus lógicas de control asociadas. En la óptica de cumplir con las unidades de conocimiento iniciales que cumplen con los Fundadamentos para el Desarrollo de Software (*Software Development Fundamentals (SDF)*) en su integración con las estructuras de datos mencionadas y el diseño de algoritmos según lo dispuesto por la ACM/IEEE-CS, específicamente en el documento *Computing Science Curricula 2013. The Joint Task Force on Computing Curricula Association for Computing Machinery IEEE-Computer Society*. Es de la mayor importancia aclarar que a pesar de que se da por referencia bibliográfica un año relacionado con el documento de la ACM/IEEE-CS, los autores enfatizan que los conceptos que se exponen en este libro serán vigentes hacia un futuro, porque si la teoría de números es a la Matemática como la teoría de la lógica algorítmica es a la Algoritmia, entonces la Matemática y la Algoritmia tienen el futuro asegurado, independientemente de la tecnología.

La organización del capítulo quinto corresponde a la lógica algortímica de control modular; entendido el módulo como una pieza de algoritmo esencial, compuesta a su vez por estructuras lógicas básicas, en las cuales se desarrollan los conceptos de funciones, parámetros y subrutinas. Lógica modular tiene el objetivo de proporcionar al lector los fundamentos necesarios para la construcción de algoritmos con lógicas de control autónomas, cuya integración permiten el desarrollo de lógicas algorítmicas más complejas, útiles en el desarrollo de software del mundo real. Lógica integrada que de lo simple a lo complejo permite la interrelación de lógicas de entrada/salida/proceso (o lógicas básicas) para conformar funciones o subrutinas (algoritmos primarios con acciones autónomas) gobernadas por un algoritmo principal.

En el capítulo seis se plantean unas situaciones de la vida real que se resuelven mediante algoritmos. Estas situaciones problemas son de mayor dificultad y por lo tanto se dejan al final del libro. Es opcional la lectura de este capítulo. Si el lector desea profundizar en el tema, se le recomienda leer este capítulo.

Con relación a los capítulos tres, cuatro y cinco, es de la mayor importancia resaltar que la construcción de algoritmos con base en las lógicas básicas (significado que para algunos autores se asocia al término "pseudocódigo"), dichos algoritmos pueden ser codificados en cualquier lenguaje de programación, como es el caso de

Java, C, C++, entre otros. Igualmente, los conceptos desarrollados, al ser independientes tanto del desarrollo de los lenguajes de programación como del avance de las máquinas de computación, en su visión de futuro teórico-práctica nunca perderán su relevancia, por cuanto están soportados por estructuras lógicas.

Finalmente, se presentan las conclusiones y la bibliografía empleada como soporte a la conceptualización del tema relacionado con la lógica algorítmica como base a la generación de la lógica de programación.

## Bibliografía

Brassard, G. & Bratley, P. (1997). *Fundamentos de Algoritmia.* Madrid: Prentice Hall.

Computer Science Curricula (2013). The joint task force on computing curricula association for computing machinery IEEE-Computer society.

# Capítulo 1

# Lógica de programación

## 1.1 Concepto de Lógica

El desarrollo de la Informática como ciencia que estudia el tratamiento de la información a través de máquinas de procesamiento electrónico de datos al servicio de una sociedad digital y global se basa tanto en la tecnología como en la capacidad racional humana, con el fin de transformar un conjunto de hechos a partir de los cuales se logra deducir a través de la razón humana un conjunto de resultados, que al ser valorados en su condición de verdad pueden ser lógicamente correctos o incorrectos. Lo anterior implica, para el avance informático, tanto el desarrollo tecnológico como el desarrollo de las formas de razonamiento humano.

El ser humano racional tiene su base biológica en el cerebro, como órgano que permite a través del lenguaje utilizar los métodos y principios para distinguir el razonamiento correcto del incorrecto. La capacidad de razonamiento humano se basa en un conjunto de reglas o formas válidas de deducción o inferencia, utilizadas por la civilización griega. El conjunto de reglas mencionado conformaba un sistema de razonamiento, y los griegos "llamaron a este sistema de razonamiento 'Lógica'" (Dean, 2003, p. 1). El filósofo griego Aristóteles fue el primero en referirse a la noción de lógica en el entendido de que los argumentos derivados de la razón son considerados como útiles para lograr la verdad en una determinada ciencia. Luego el término "lógica", que etimológicamente se deriva del latín *lógica*, que a su vez proviene del griego antiguo *logike*, es usado para referirse a un sistema de razonamiento formal. Es este sistema de razonamiento formal el que se utiliza en la construcción de algoritmos; y es precisamente el sistema de reglas que componen un algoritmo lo que se codifica y se ejecuta en un computador para derivar de dicha ejecución resultados que, en síntesis, se convierten en información útil al usuario.

Los sistemas de razonamiento a través del lenguaje han evolucionado de un razonamiento puramente verbal denominado "lenguaje natural" a razonamiento simbólico o lógica simbólica, justificado por el hecho de que

> ... en la lógica simbólica la información es representada usando letras y símbolos, similares al Álgebra. La lógica simbólica posibilita un cierto grado de automatización del razonamiento; sin embargo, su motivación original fue el deseo de ser capaz de solucionar un problema lógico a través de "cálculos", justamente como la respuesta a un problema numérico puede ser calculado usando aritmética. Por esta razón, la lógica simbólica está cerradamente asociada con computadores... (Dean, 2003, p. 2)

Luego, con base en lo anterior, la lógica es una rama de la filosofía que estudia de manera formal las deducciones válidas que se derivan de un sistema de razonamiento fundamentado en un conjunto de reglas. Si el sistema de razonamiento mencionado se expresa en un lenguaje matemático, recibe el nombre de "lógica matemática"; en el caso de que el sistemas de razonamiento utilice un lenguaje simbólico y un conjunto de reglas de inferencia recibe el nombre de "lógica simbólica"; y es precisamente esta última la que a través de los algoritmos conformados por estructuras lógicas ha permitido el desarrollo de la informática. El concepto de lógica simbólica antes mencionado, en su intento por alcanzar una automatización utiliza la algoritmia, a cuya ciencia pertenece el concepto de algoritmo.

## 1.2 Concepto de algoritmo

### 1.2.1 Síntesis histórica del desarrollo de los algoritmos

Los algoritmos relacionados con la lógica, y esta, a su vez, teniendo relación tanto con la matemática como con el lenguaje, no pueden ser aislados en su desarrollo histórico, por lo tanto, de la historia de los sistemas numéricos y de los sistemas de alfabetos de soporte a un lenguaje. Las primeras construcciones de algoritmos se remontan a la civilización babilónica (2000-600 a.C.), y están relacionadas con el campo matemático, particularmente en la resolución de ecuaciones y en la utilización del sistema numérico sexagesimal (Base 60 ($b_{60}$)); sistema heredado de los sumerios (2500 a.C.), cuya ubicación geográfica es el valle del río Euphrates (Irak, sur de Baghdad); sistema que se fundamentó en signos especiales para los números 1, 10, 60, 600 y 3600, de los cual se puede concluir que trabajaban tanto en base 10 como en la base sexagesimal.

Es importante destacar que "... Al comienzo del segundo siglo antes de Jesucristo, los babilonios desarrollaron un sistema de numeración, el cual se mantiene en nuestros días en el uso de los minutos y segundos para el tiempo y en la medida de un ángulo" (Chabert & Barbin, 1999, p. 11).

La necesidad del lenguaje en el proceso de comunicación dentro de las sociedades se hizo inicialmente a través de la descripción de acciones cotidianas utilizando figuras (3000 a.C.), para posteriormente ser simplificada la representación de acciones

en símbolos bases para la generación de alfabetos, de lo cual se derivó el lenguaje escrito. "El primer alfabeto verdadero apareció cerca de 1700 a.C. en el Este del Mediterráneo" (Wear et al., 1991, p. 7).

La contribución de la antigua civilización egipcia en el desarrollo de los algoritmos se basa en su sistema decimal, representado a través de símbolos que siendo escritura jeroglífica sirvió para representar palabras (2000 d.C.); "los egipcios también usaron unidades fraccionarias" (Chabert & Barbin, 1999, p. 15), y de esta época data la representación de fracciones como 2/3 y 1/3, a la cual se le llamó 'la pequeña mitad'.

Teniendo en cuenta que hay muchos algoritmos para la multiplicación, se destaca que la multiplicación en forma de tabla, como se conoce actualmente, tuvo su génesis en diferentes tiempos en la China, la India, el mundo árabe y la zona europea. Las evidencias de las tablas de multiplicar en China datan del año 1450, utilizadas a nivel financiero en la provincia de Zhejiang; por su parte, en Europa la más reciente evidencia del algoritmo de multiplicación aparece "... en un manuscrito en latín, alrededor del año 1300 en Inglaterra en la región de Edward II" (Chabert & Barbin, 1999, p. 25).

El desarrollo del lenguaje a través de la teoría de alfabetos después de su primera aparición se complementó con la creación de los alfabetos etrusco y griego alrededor de los años 800-700 d.C., llegando hasta el antiguo alfabeto romano en el año 600-100 d.C., y finalmente el nuevo alfabeto romano durante los años 400-1500 (a.C.).

En el desarrollo algorítmico de las operaciones aritméticas es importante destacar el ábaco chino por su relación con los cálculos de las operaciones suma, resta y la multiplicación, ubicado históricamente en la segunda mitad del siglo XVI. Las operaciones aritméticas solo fueron posibles con el desarrollo de los sistemas de numeración, cuyo desarrollo fue paralelo al alfabeto, por la necesidad de la sociedad de tener un sistema de conteo; así, el sistema más comúnmente utilizado fue el sistema decimal o en base 10 ($b_{10}$). En el sistema mencionado, y teniendo como base la unificación de precios y medidas de la antigua dinastía Quin en China (210 d.C.), se utilizaron las decenas, centenas y unidades de mil como cantidades posicionales del sistema decimal, lo cual hizo posible la aparición de fracciones decimales. El sistema de numeración binario o base dos ($b_2$) tiene como representante en la aritmética binaria a

> Leibniz, quien escribió su memoria sobre La Explicación de la Aritmética Binaria en 1703, pero estuvo por más de quince años interesado en un sistema de numeración, el cual fue para él un modelo de lo que llamó una característica universal. (Chabert & Barbin, 1999, p. 11).

El sistema binario tuvo un sistema posicional generalizado construido por el mismo Leibinz en 1666; pero se debe tener en cuenta que realmente el primer sistema de posicionamiento digital se le debe a Thomas Harriot (1560-1621), cuya obra no fue publicada y Leibniz lo realizó de una forma independiente. Es importante destacar que con la creación del sistema binario fue posible reducir la lógica a operaciones mecánicas; y siendo reducida la lógica a estas operaciones con el nacimiento de la

máquina para la realización de sumas llamada “máquina de Pascal”, habiendo Leibniz inventado una máquina para la ejecución de las cuatro operaciones aritméticas, este tipo de máquinas precedió a las de computación moderna, entre las cuales se encuentra la de Charles Babbage, cuya identidad es la ‘máquina de diferencias’, inventada por este en 1822. De forma paralela, es importante destacar los siguientes eventos en la evolución de la matemática relacionada con la lógica binaria: el tratamiento matemático a través de los conceptos del cálculo diferencial e integral se le acreditan a Leibniz y Newton entre 1675 y 1687. La introducción del concepto de función matemática se le debe a Euler, quien alrededor de 1748 formalizó los conceptos de la matemática analítica. La definición del Álgebra Lineal se le debe a Johan Carl Friedrich Gauss (1777-1855), a quien también se le atribuyen importantes contribuciones en los campos de Teoría de Números, Estadística, Geofísica y Electrostática, entre otros campos, dentro de los cuales se encuentran importantes algoritmos, como son los casos del cálculo del promedio y la desviación estándar. Adicionalmente, la creación de las llamadas retículas booleanas o el Álgebra de Boole se le debe a George Boole (1815-1864), de fundamental importancia para el desarrollo computacional.

La necesidad que ha tenido la sociedad de llevar un registro de sus avances en términos cuantitativos y cualitativos hizo posible el desarrollo de la teoría de alfabetos y lenguajes de comunicación entre humanos, desde los antiguos papiros egipcios, pasando por la invención de la imprenta (1398-1400) por Johannes Gutemberg y llegando hasta los actuales sistemas de información basados en el computador. Igualmente, el registro de las acciones humanas permitió con base en el avance de los sistemas numéricos, y particularmente sobre el fundamento del sistema binario, el desarrollo de la computación actual.

Teniendo en cuenta los desarrollos en el campo del lenguaje y la matemática, se le atribuye a Howard Aiken y Grace Hopper la creación del primer computador, identificado como el *Harvard Mark I Computer*, hacia 1944; y posteriormente, la invención del computador de tubos al vacío, llamado el ENIAC I, en 1946, a cargo de Jhon Presper Eckert y John W. Mauchly, a quienes también se les acredita la creación del computador UNIVAC en 1953. En el mismo año, la compañía International Business Machines (IBM) entra a participar en el desarrollo de la computación con la fabricación del computador IBM 701 EDPM; y en 1954 se crea el primer lenguaje programable en un computador de alto nivel llamado FORTRAN, a cargo de John Backus y la compañía IBM.

Los eventos o acciones más importantes en el desarrollo de la algoritmia desde la Edad Antigua hasta la Edad Moderna son los siguientes, teniendo en cuenta que dichos eventos están necesariamente ligados con el desarrollo de la Matemática, la Ciencia de la Computación y la Informática:

- (3000 a.C.): creación de símbolos para representativos de las acciones humanas.
- (2000 a.C.): los babilonios inventan el sistema sexagesimal.
- (1700 a.C.): creación de las letras para la representar sonidos individuales.

- (1600 a.C.): Demócrito de Abdera encontró la fórmula para calcular el volumen de una pirámide.
- (750-850 a.C.): Al Khowarismi inventó las reglas de las cuatro operaciones básicas de la aritmética, época en que se acuño el término "algorism".
- (408 a.C.): Eudoxo inventó el método para demostrar de una forma rigurosa el cálculo de áreas y volúmenes utilizando aproximaciones sucesivas.
- (300 a.C.): Euclides inventó el algoritmo para calcular el máximo común divisor de dos números, n y m. MCD (n.m).
- (222 a.C.): a Apolonio de Pergeo se le debe el tratado sobre las funciones cónicas y los nombres de la elipse, la parábola y la hipérbola.
- (1170-1250 d.C.): Leonardo de Pisa o Leonardo de Fibonacci generó el algoritmo para la sucesión de los números de Fibonacci, que son 1, 1, 2, 3, 5, 8, 13, ...
- (1550-1617 d.C.): John Napier descubrió los logaritmos.
- (1596-1650 d.C.): René Descartes creó la Geometría Analítica.
- (1623-1662 d.C.): Blaise Pascal inventó la rueda de Pascal o pascalina, considerada una de las primeras calculadoras.
- (1642-1727 d.C.): Isaac Newton desarrolló el teorema del Binomio o el Coeficiente Binomial.
- (1646-1716 d.C.): Gottfried Leibniz creó el concepto de arreglo o matriz; porque fue el primero en organizar los coeficientes de un sistema de ecuaciones lineales en forma de filas y columnas.
- (1736-1813 d.C.): Joseph Louis de Lagrange en su obra *Mecánica Analítica* creó las ecuaciones de Lagrange para sistemas dinámicos.
- (1749-1827 d.C.): Pierre Simón Laplace escribió la obra *Teoría Analítica de las Probabilidades* (1812).
- (1815-1864 d.C.): En la obra *Investigación de las Leyes del Pensamiento* (1854) George Boole su hace un aporte estructural; creó el Álgebra booleana (1848).
- (1862-1943 d.C.): David Hilbert introdujo la noción del espacio de Hilbert, que es uno de los pilares del análisis funcional.
- (1903-1957 d.C.): John von Neuman contribuyó con el desarrollo de los primeros computadores electrónicos.
- (1903-1995 d.C.): Alonzo Church, uno de los más importantes teóricos de la Ciencia de la Computación en el siglo XX, a quien se le debe el concepto del Cálculo de Lambda, definió la idea de "Calculabilidad Efectiva".

- (1912-1954 d.C.): Alan Mathison Turing introdujo el concepto de la máquina de computación abstracta denominada la Máquina de Turing.
- (1924-d.C.): Donald L. Shell creó el algoritmo de ordenamiento Shell (1959).
- (1938-d.C.): Donal Knuth creó la obra *El arte de Programar Computadores* e hizo uno de los mayores aportes en la Ciencia de la Computación, realizando trabajos importantes en las áreas de Análisis de Algoritmos y Compiladores.

Adicionalmente, se presentan algunos de los científicos que mayores aportes han realizado:

- Wirth Niklaus (1934-d.C.), científico suizo de la computación, quien se desempeñó como jefe de Diseño de los lenguajes de programación identificados como Euler, Algol W, Pascal, Módula I, Módula II y Oberón.
- Charles Leiserson (1953-d.C.), científico informático de los Estados Unidos cuyos trabajos en algoritmia han sido relevantes, particularmente en las áreas de la computación paralela y distribuida.
- El algoritmo de Sethi-Ullman (1970-d.C.), llamado finalmente el algoritmo de Ravi Sethi y Jeffrey D. Ullman, crearon en el área de compiladores el algoritmo para traducir los árboles del análisis sintáctico a códigos de máquina.
- Michael R. Garey y David S. Johnson autores del importante libro *Computers and Intractability. A guide to the Theory of NP-Completeness*, el cual contiene la teoría para el tratamiento de problemas NP-Completos y clasificación de los tipos de problemas NP-Completos en las áreas de la Ciencia de la Computación relacionadas con Teoría de Grafos, Diseño de Redes, Autómatas y Teoría de Lenguajes entre otras.

Concluida la síntesis histórica del desarrollo de los algoritmos, la siguiente sección desarrolla el concepto de algoritmo.

### 1.2.2 Algoritmo

Un *algoritmo* es un conjunto finito de reglas bien definidas en su lógica de control que permiten la solución de un problema en una cantidad finita de tiempo. En la resolución del problema con las reglas mencionadas, el algoritmo realiza un conjunto de pasos cuya ejecución para dar la solución del problema puede ser ejecutada manualmente, mecánicamente o utilizando una máquina de procesamiento electrónico de datos.

Al estar conformado por un conjunto finito de reglas, el algoritmo tiene: i) una(s) regla(s) de entrada(s) al algoritmo o punto inicial del algoritmo; ii) una o un conjunto de reglas intermedias (reglas de proceso o de cálculos); iii) un conjunto o una regla de

finalización del algoritmo, que aseguran su terminación, punto en el cual se obtiene una respuesta útil derivada de la ejecución del conjunto de reglas del algoritmo, y iv) exactitud, en el proceso lógico de organización de las reglas a fin de dar respuesta al usuario, o lo que es equivalente: "Un algoritmo se espera resuelva un problema" (Joyanes, 2008, p. 13) de una forma correcta. Si el algoritmo tiene un conjunto infinito de reglas, implicaría que su construcción o está por terminarse pero el número de sus reglas no es mensurable o su construcción es tan compleja que requeriría una escritura infinita de reglas, lo cual no es útil para el usuario en términos de la resolución de problemas.

El algoritmo en la ejecución del conjunto de reglas utiliza un tiempo finito entre la regla de entrada del algoritmo y la ejecución de su regla final; lo cual implica que el tiempo de demora del algoritmo es mensurable; luego, si el tiempo de demora del algoritmo no se puede acotar, o lo que es equivalente, es infinito, el algoritmo y sus reglas no tendrían un sentido útil para el usuario, por cuanto este no puede esperar un tiempo infinito en la solución del problema factible de ser tratado a través de la teoría de algoritmos.

Luego, si $R_i$ es el conjunto de reglas lógicas que conforman un algoritmo, acotadas las reglas lógicas del algoritmo por $k$ $(1 \leq i \leq k)$, y sea $n$ el número de datos de entrada del algoritmo, entonces el tiempo de demora del algoritmo en función del número de datos de entrada se identifica matemáticamente por $T(n)$. Las $R_i$ reglas que componen el algoritmo deben estar bien definidas y corresponder a una lógica de control algorítmico. Esta lógica de control tiene unos constructores lógicos o estructuras básicas para la construcción de la lógica de control algorítmico. Estas estructuras, al estar bien definidas, permiten la obtención de resultados (cálculos aritmético/lógicos) y, consecuentemente, la solución del problema. La organización de las estructuras básicas mencionadas crean dependencias funcionales entre sí; en tal sentido, no se pueden obtener unos resultados sin los datos de entrada (materia prima de la solución de un problema), como tampoco se tendrían resultados sin reglas que conformen el proceso de transformación de los datos mencionados en información de resultado para el usuario.

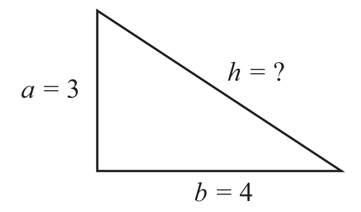
---

Ejemplo 1.1 Cálculo de la hipotenusa de un triángulo rectángulo

---

Sean $a$ y $b$ los lados de un triángulo rectángulo, cuyos valores son $a = 3$ y $b = 4$, diseñar un conjunto de reglas algorítmicas para calcular la hipotenusa del triángulo.

*Análisis*:



Con base en el teorema de Pitágoras, la hipotenusa de un triángulo rectángulo es igual a $h^2 = a^2 + b^2$.

*Solución*:

Primero se necesita una regla mediante la cual obtengamos los datos de entrada. Entonces $R_1$ es igual a obtener $a$ y $b$.

Posteriormente se hace necesario una segunda regla que calcule el valor de la hipotenusa, de acuerdo con la ecuación de cálculo. Luego, como $h^2 = a^2+b^2$, entonces $h = \sqrt{a^2 + b^2}$, por lo tanto: $R_2$ = Calcule el valor de la hipotenusa $h$ que es igual a la raíz cuadrada de $a$ elevado al cuadrado más $b$ elevado al cuadrado ($h = \sqrt{a^2 + b^2} = \sqrt{3^2 + 4^2} = \sqrt{25} = 5$).

Adicionalmente es necesaria una tercera regla que muestre el resultado del cálculo. Luego, $R_3$ = Muestre el valor de la hipotenusa ($h$) calculada.

Finalmente, organizamos de una forma lógica las reglas definidas anteriormente. Es decir, su ejecución manual (la cual es equivalente a lo realizado por un computador) es $R_1 \rightarrow R_2 \rightarrow R_3$. Si y solo si disponemos la secuencia de reglas de esta forma logramos el cálculo del valor de la hipotenusa. Por lo tanto, la secuencia de reglas $R_1$, $R_2$, $R_3$ es un algoritmo. La sencillez en el cálculo de la hipotenusa con los datos dados contrasta con la complejidad de su cálculo cuando los lados $a$ y $b$ del triángulo rectángulo tienen una longitud de uno, cuyo cálculo es

$$h = \sqrt{a^2 + b^2} = \sqrt{1^2 + 1^2} = \sqrt{2} = 1.41421356237309\ldots$$

Lo anterior induce que de acuerdo con los primeros datos de entrada ($a = 3$ y $b = 4$), la respuesta del algoritmo es exacta; pero no existe un algoritmo que pueda dar en notación decimal una respuesta exacta de la $\sqrt{2}$, por cuanto su secuencia de decimales es irrepetible e infinitamente larga.

La dependencia funcional de las reglas dadas induce que $R_3$ depende para dar los resultados funcionalmente de $R_2$; a su vez, $R_2$ depende funcionalmente de $R_1$ para poder calcular los valores de la hipotenusa en función de la raíz; pero finalmente el resultado o información de salida, en cuanto a la ejecución de la primera regla ($R_1$), también depende de los datos de entrada.

La $R_i$ reglas mencionadas en la resolución del problema dado son las estructuras lógicas básicas que hacen que el cálculo de valor de la hipotenusa sea posible; o lo que es equivalente, son los constructores lógicos de entrada u obtención de datos ($R_1$), cálculo o proceso de datos ($R_2$), y finalmente salida de la información de la hipotenusa ($R_3$).

# 1.3 Concepto de sistema

## 1.3.1 Sistema

El concepto de *sistema* se desarrolla en el marco de la Teoría General de Sistemas (TGS). La TGS centra su fundamento en un enfoque multidisciplinario que sirve para estudiar las propiedades de las entidades del universo. Una "entidad" es algo identificable cuyas características se pueden definir en términos de la racionalidad humana y es entendible por sus propiedades para la comprensión del hombre. Este término fue acuñado en el siglo XX con el trabajo de Ludwing von Bertalanffy, a cuyo nombre se deben las investigaciones de sistemas abiertos. Con base en lo anterior, una entidad es una organización (el Estado, la universidad, la familia) que conforma una estructura social. El funcionamiento de las estructuras sociales se basa en personas (hombres y mujeres con consciencia racional e inteligentes) que dentro de la organización desempeñan unas funciones de una forma organizada según la estructura de la organización; y en tal sentido son funcionarios que dentro de la estructura del Estado realizan funciones de gobierno y defensa de un país; o dentro de la organización universidad hacen el trabajo académico, administrativo y de investigación del colectivo que se une en la diversidad; o en la familia desempeñan funciones de rol de padre y de madre, quienes conjuntamente guían a los hijos.

Las organizaciones, por lo tanto, dependen de las personas para su funcionamiento; y a su vez, las personas, como seres animales racionales, dependen de las plantas para la supervivencia de sus células biológicas, siendo la célula un sistema abierto.

Se identifica un ordenamiento jerárquico con complejidad decreciente en las entidades anteriormente relacionadas, lo cual se concreta en unidades de información, a saber:

1. Sistema Social: La entidad "organización" tiene un NIT dentro de la sociedad. Ej. NIT de la Universidad.

2. Sistema Humano: Las entidades "el hombre" y "la mujer" se identifican por su nombre en cualquier sociedad, lo cual se complementa con la cédula de ciudadanía, los números del seguro social o los números de los pasaportes. Ejemplo: Pedro Jesús Blanco, con cédula de ciudadanía 72´425.777; y Debra Kimberly Hausser, con número de pasaporte AFDK-1418-20.

3. Sistema Genético Vegetal: La entidad "planta", la cual puede ser clasificada según varios sistemas de clasificación, como son los casos de los sistemas de clasificación de angiospermas de APG[1]. Ejemplo: clasificación del olivo o aceituno como planta oleaginosa.

[1] "Angiosperm Phylogeny Group" (APG). Disponible en:
http://es.wikipedia.org/wiki/Clasificaci %C3 %B3n_de_los_organismos_vegetales

4. Sistema Celular: La entidad "célula", la cual puede ser clasificada en células animales y vegetales. Ejemplo: célula eucariota, aquella que teniendo un núcleo, contiene su información genética en "una doble membrana, la envoltura nuclear la cual delimita un núcleo celular".[2]

Las entidades anteriormente relacionadas tienen unidades de información, las cuales se pueden expresar tanto sintáctica como semánticamente a través de números y letras. Ejemplo: código numérico de un alumno de una universidad: 20110213044, cuya semántica se puede interpretar como 2011, año de ingreso, 02, ingreso en el segundo semestre, 13 es el número de la unidad académica a la cual pertenece o Programa de Ingeniería Industrial, y finalmente 044 es el alumno posicionado 044 en la secuencia de ingreso al programa mencionado.

Teniendo en cuenta el marco de la Teoría General de los Sistemas, se intentará concretar algunas definiciones del término "sistema".

Con base en los objetos como partes componentes de un sistemas y los atributos asociados a los objetos, Hall (Johansen, 1992, p. 55)

> (...) define un sistema como un conjunto de objetos y sus relaciones, y las relaciones entre los objetos y sus atributos.

Una definición a nivel genérico del concepto de sistema es:

> ... Cualquier o alguna cosa del mundo real capaz de ser vista como un sistema. Cualquier referente de sistema debe ser capaz de ser representado como un todo o como una colección de partes relacionadas. Cualquier parte de un sistema debe ser capaz de ser vista como un sistema en sí mismo. Igualmente, un sistema debe ser capaz de ser absorbido por un sistema mayor como una de sus partes. (Myers & Kaposi, 2004, p. 38)

Desde el punto de vista del tipo de sistema social, económico, ambiental, técnico o informático:

> ... Sistema se refiere a sistemas técnicos tales como un sistema de transmisión electrónica. Algunas veces se refiere un sistema como interactuando con componentes, algunas de las cuales no son técnicas. Sistema dirigido principalmente en la interacción de componentes institucionales y técnico... Sistema entonces significa la interacción de componentes de diferentes formas, tales como técnicas o institucionales, también como diferentes valores. (Coutard, 1999, p. 248)

El término "sistema" puede ser referido también a la confiabilidad en procesos predictivos, cuando el sistema está integrado por componentes; una de los cuales puede ser un conjunto de algoritmos que soportan el software de un proceso determinado, y se expresa el concepto de sistema como

[2]Célula eucariota. Disponible en: http://es.wikipedia.org/wiki/C %C3 %A9lula_eucariota

> Cualquier equipo / producto compuesto de unidades de subsistemas / componentes, los cuales deben ser especificados por sus configuraciones funcionales y límites. Esta información ayudará a hacer los diagramas de bloques de los productos/equipos funcionales para evaluar la confiabilidad. La falla de los componentes en un bloque hará que el sistema falle dependiendo de su configuración. (Mishra & Sandilya, 2009, p. 117)

Los diferentes enfoques del término "sistema" hace que su variabilidad conceptual, por su polisemia, sea difícil de definir concretamente; luego, el concepto de sistema que se maneja en este libro, dentro de un enfoque informático soportado por la teoría de algoritmos, es el siguiente:

Un sistema es un conjunto de entidades que interactúan entre sí de una forma organizada, con el fin de desempeñar una función y generar una información útil al usuario, dentro de unos límites de operación y funcionalidad.

Se resaltan en la definición las siguientes categorías conceptuales: *i*) Conjunto de entidades: las entidades son las partes componentes del sistema, lo cual genera los conceptos de supra sistemas, sistema y subsistemas. Ejemplo: Suprasistema "carro"; integrado por los sistemas mecánico, neumático y eléctrico; a su vez, el subsistema mecánico, compuesto por el motor, la dirección, el acelerador y los frenos del carro. *ii*) Interacción de entidades: la entidades dentro de los sistemas generan procesos de conectividad y comunicación; dichos proceso hacen que los sistemas sean estáticos o dinámicos. Una estructura geométrica, como es el caso de un cubo sobre una mesa en la tierra, por la gravedad tiende a ser un sistema estático, y una estructura esférica sobre un plano inclinado en un ángulo de 45 grados tiende a ser dinámica. *iii*) Interacción organizada: la organización es contraria al caos; y es precisamente la interacción de una forma controlada de las partes de un sistema la que genera información. En el suprasistema "carro" mencionado, el punto de apoyo del carro es el sistema neumático; ahora, el sistema neumático ha de interactuar de una forma controlada con el sistema eléctrico, el cual permite el arranque del carro, y a su vez, dichos sistemas deben interrelacionar de una forma organizada con el sistema mecánico, generando así el sistema de desplazamiento de vehículo. Contraria a la organización antes mencionada, si el carro en su sistema de desplazamiento, estando en bajada, se arranca en una quinta velocidad y a una máxima aceleración, el sistema puede salirse de control. *iv*) Generar información útil al usuario: son los resultados del sistema para el cual fue concebido o por lo cual interactúan sus partes empleando unidades energéticas y espacio temporales, entre otras. En tal sentido, dentro del sistema de desplazamiento del carro, cuando el vehículo está en movimiento, la utilidad del sistema es la movilidad para el usuario de un punto geográfico inicial a un punto final; pero, a su vez, se generan en el desplazamiento otras unidades de información, tales como: la velocidad del carro en su desplazamiento, número de kilómetros/hora; el consumo de gasolina número de galones/kilómetro; la temperatura del motor, entre otras informaciones. *v*) Contorno del sistema: de acuerdo con los resultados de información, el sistema debe funcionar en un entorno determinado o área de operación; ello hace que el sistema tenga unos límites preestablecidos, contorno que diferencia un sistema de otro, pero, a su vez, es el punto de interacción

entre subsistemas; así, el sistema "neumático" (llantas) tiene su contorno, muy diferente al sistema mecánico (motor); pero ambos, llantas y motor, interactúan para darle al carro el desplazamiento. Algorítmicamente, es necesario primero accionar el sistema eléctrico del carro, que enciende el motor, para poder accionar a través del eje de transmisión del carro la energía que da fuerza de desplazamiento a las llantas, para que finalmente estas sean controladas por el acelerador y el freno del carro.

Una de las características fundamentales de la definición de sistema es la generación de información. La información es una cantidad mensurable que debe ser medida en una escala de valoración cuantitativa o cualitativa. La valoración de la información requiere de un procesamiento de las unidades primarias de información, que son los datos; por lo tanto, el concepto de información se asocia también a un conjunto organizado de datos derivados de un sistema en operación. Organizar los datos y evaluarlos requiere necesariamente de un sistema de posicionamiento o de clasificación de datos y de un sistema de valoración de los mismos. En el sistema digital, los números en el intervalo cerrado [0;9] son base para los sistemas de codificación; un sistema de codificación digital de $k$ dígitos ($k = 4$) y de $n$ códigos ($n = 5$) requiere para su organización posicionamiento; en tal sentido, los códigos digitales representativos de las referencias de partes de una compañía de producción especificados como 4584, 9872, 3210, 1400, 7777 son fácilmente valorables en la escala digital por su posicionamiento, de lo cual se puede concluir que la menor referencia es la 1400, y la mayor referencia es la 9872, soportada la conclusión por el sistema de posicionamiento de las referencias en su orden: 1) 1400, 2) 3210, 3) 4584, 4) 7777 y 5) 9872. Las referencias fueron fácilmente organizadas para la compañía referidas a un sistema de inventarios; pero, si tanto los códigos del sistema de codificación aumentan, supuesto $k \geq 10$ dígitos por referencia, y el número de dígitos aumenta, supuesto $n \geq 1$ millón, entonces la organización de los datos vía a convertirlos en información (referencias clasificadas) útil al usuario requiere necesariamente de un sistema de procesamiento electrónico de datos basado en un computador; para ello se requiere entonces construir un algoritmo que ordene las referencias del sistema de inventarios mencionado.

## 1.4 Concepto de algoritmo en el marco de la Lógica

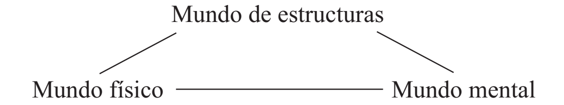
Teniendo como fundamentos que la lógica es entendida como un conjunto de reglas que conforman un sistema de razonamiento y, a su vez, siendo el algoritmo un conjunto finito de reglas bien conformadas en su lógica de control y factibles de ser automatizadas mediante un lenguaje de programación, el análisis de la relación entre algoritmo y lógica se basará en la Teoría de la Información; justificado por el hecho de que es precisamente la operación de la lógica de control del algoritmo la que permite el procesamiento de los datos para generar información y, en últimas, conocimiento útil en el contexto de producción de un sistema informático.

La información es un recurso de decisión en la sociedad digital, y sirve para desarrollar el conocimiento humano. El desarrollo histórico del lenguaje como medio de comunicación entre humanos y vía de interacción en la relación hombre-computador, ligado al desarrollo histórico de la teoría de alfabetos, ha hecho posible que los fenómenos del mundo real sean representados en el mundo imaginario del hombre mediante símbolos. Un símbolo, en el contexto del lenguaje, es una secuencia de caracteres con una sintaxis y una semántica asociada que pertenecen a un alfabeto. Así, la secuencia 1010 es una sarta de dígitos binarios cuyos caracteres pertenecen a un alfabeto definido como el conjunto de imágenes pictográficas llamadas "símbolos, vocabulario, o cualquier conjunto finito de símbolos" (Hopcroft & Ullman, 1969, p. 1); secuencia de bits 1010 que pertenecen entonces al alfabeto binario o en base dos igual a $b_2 = \{0, 1\}$ en su estructura sintáctica; y en su semántica o significado, de acuerdo con el sistema posicional binario, corresponde al número decimal $(12)_{10}$. Con base en lo anterior, la palabra "algoritmo" es una sarta de nueve símbolos, compuesta por cuatro vocales y cinco consonantes que pertenecen al alfabeto latino.

La representación en el mundo real de un algoritmo a nivel fáctico es un conjunto de reglas lógicas. Las reglas lógicas que componen el algoritmo forman un imaginario en la mente del sujeto o alumno. En el imaginario del sujeto, es la organización de la lógica propia de la mente de este la que permite simbolizar la solución de un problema determinado utilizando la teoría de algoritmos. En tal sentido se unen dos sistemas de operación: por un lado, las reglas lógicas que sirven para construir los algoritmos y, por otro lado, las reglas lógicas (desarrolladas o no) presentes en la mente del sujeto. De resultar, si el sujeto tiene una estructura de pensamiento formal, que le sirve para pensar en complejos, es más fácil para el aprendiz de algoritmos construir rápidas y efectivas soluciones algorítmicas. Ello indica que la lógica formal tiene una estrecha relación con la lógica algorítmica; y a mayor nivel de desarrollo de la lógica formal del sujeto se generará un mayor nivel de estructuración formal en las reglas algorítmicas y, consecuentemente, una mayor posibilidad de automatización de un problema a ser solucionado con base en la algoritmia y la computación.

El paso del mundo real (reglas algorítmicas) al mundo imaginario (lógica del constructor del algoritmo) del sujeto produce como consecuencia la representación simbólica de la solución del problema en su mente a través de su lógica. La representación simbólica del algoritmo en la mente del sujeto no es aún la solución del problema a nivel informático utilizando la teoría de algoritmos; hasta ahora se ha cubierto la etapa de la solución del problema a través de la lógica humana, cuya representación simbólica es la lógica algorítmica. La lógica algorítmica, concretada en un algoritmo, necesita la utilización de un lenguaje de programación, con el fin de que la representación del algoritmo pueda codificarse en un conjunto de símbolos que pertenecen no ya a un lenguaje humano, sino a un lenguaje de programación, que puede ser ejecutado en una máquina de procesamiento electrónico de datos o computador. Este es precisamente el punto en el cual la lógica humana al interactuar con la lógica algorítmica se convierte en lógica de programación, justificando así la relación entre algoritmo y lógica; y llegando así a convertirse en lógica de programación.

El análisis anterior está soportado por la llamada "Triada existencial del mundo" (o estructura global del mundo), que tiene la forma




(Burgin, 2009, p. 60).

El mundo de las estructuras contiene sus tipos, tales como: estructuras algorítmicas, estructuras de datos, estructuras de lenguajes de programación, entre otras, en un contexto informático. Con el conglomerado del mundo de estructuras se modela el mundo físico; o sea, en el mundo físico se encuentra tanto el problema que se va a solucionar como las reglas algorítmicas, ambas son fácticas. La triada se cierra cuando las estructuras algorítmicas solucionan el problema al pasar por el mundo mental del alumno en su lógica humana racional, regresando al mundo de las estructuras organizadas, donde se convierten en estructuras lógicas algorítmicas. El calcular el problema de la función factorial ($n!$) en su raíz pertenece al mundo de las estructuras matemáticas; ello indica que el factorial es una función representada por !, cuyos elementos matemáticos son el 1 como módulo de la multiplicación y el $\times$ como operador de la operación de multiplicación. Con base en la estructura matemática de soporte a la función mencionada, para calcular la función del mundo real $n!$ (de aplicación en el coeficiente binomial ($n!/(k!(n-k!))$ utilizando algoritmos, en el mundo real se encuentran las estructuras de control algorítmico; estas estructuras indican en la realidad que si el sujeto en su mundo mental organiza una secuencias de multiplicaciones desde un punto inicial (1) hasta un punto final (5) en la búsqueda de la respuesta del 5!, obtiene la respuesta de la función para dicho valor, o sea 120; ya en este punto del resultado se regresa al mundo de las estructuras, lugar en el cual se encuentran las estructuras lógicas algorítmicas o de lógica de programación si el algoritmo se codifica. Se enfatiza que si la mente del sujeto no logra integrar las estructuras matemáticas con las estructuras de control algorítmico a través de su mundo mental, puede no tener una respuesta correcta.

Es de la mayor importancia resaltar que concordando con la triada anteriormente expuesta, en la mente del sujeto o en su mundo imaginario también está la triada "real - imaginario - simbólico" (Burgin, 2009, p. 67); luego, lo que realmente genera la lógica del sujeto o su sistema de razonamiento es la representación en símbolos algorítmicos de la resolución del problema.

Con base en los tres mundos mencionados y la triada referenciada, la estructura algorítmica (mundo físico) en la mente del sujeto (mundo mental) debe regresar al mundo de las estructuras (estructuras de control algorítmico base para las estructuras de lenguajes de programación (lenguaje C o Java)), con el fin de que el algoritmo sea codificado en un lenguaje de programación. Las reglas algorítmicas convertidas a un lenguaje de programación se identifican como un programa de computador; y este código de programación al correrse o funcionar en un computador es el que produce información; o sea, da el resultado al usuario en el sentido de que $5! = 120$.

Este proceso corresponde a la transformación de los datos en información (dato 5, que al aplicar el factorial es información 120). Esta información ya hace parte del conocimiento humano, que se puede utilizar para una determinada decisión, como es el cálculo del Coeficiente Binomial $\binom{n}{k} = n!/k!(n-k)!$, que representa el número de subconjuntos que se pueden formar a partir de un determinado conjunto.

En tal sentido, los datos son materia prima para producir información; y la información es la base para producir conocimiento. El conocimiento como tal no se genera del simple procesamiento del dato a través del algoritmo; pues en su procesamiento, el dato es influenciado no solo por los operadores aritmético/lógicos dentro del flujo de control del programa cuando se ejecuta en el computador que transforman el dato en información, sino el dato procesado que ya es información también recibe la influencia de la(s) acción(es) de la información adicional que se le agregue a los resultados dados por el programa de computador, y que completa la interpretación del usuario de la información dada por la máquina en las estructuras de conocimiento que pertenecen al usuario en la interpretación del proceso completo de transformación del dato a información y de información a conocimiento humano.

Se justifica lo anterior por el hecho de que

> Datos, bajo (acción) la influencia de información adicional, llegan a ser conocimiento (Burgin, 2009, p. 200).

Por lo tanto, únicamente si se genera información los datos son convertidos en conocimiento; más aun, si y solo si la información generada por el programa es validada en la condición de verdad a través del sistema de razonamiento del usuario, la información llega realmente a ser útil al usuario, y toda la energía gastada en la resolución del problema a través de la construcción del algoritmo y la codificación del programada no se ha perdido tanto en tiempo como en esfuerzo.

El análisis del concepto de algoritmo en su relación con la lógica conduce a identificar que la resolución de problemas factibles de ser tratados por máquinas de procesamiento electrónicas de datos o computador inicia en el sistema de razonamiento humano, pero también termina en el conjunto de reglas del razonamiento humano. Las reglas algorítmicas y de programación, que generan la lógica de programación, son solo un paso, en este caso electrónico, para lo cual el computador es una herramienta de cálculo, pero la esencia de la resolución de problema es en sí la lógica humana.

Una vez desarrollados los conceptos de algoritmo, sistema y el concepto de algoritmo en el marco de la lógica, el paso siguiente es explorar conceptualmente las máquinas de procesamiento electrónico de datos.

## 1.5 Máquinas de procesamiento electrónico de datos

### 1.5.1 Sistema Computacional

Teniendo en cuenta que un sistema es un conjunto de entidades que al interrelacionar entre sí de una forma controlada dan una respuesta de utilidad para el usuario, el sistema computacional se basa en un conjunto de partes (el computador y la red) que interactúan de una forma organizada para dar resultados al usuario. Luego, un sistema computacional basado en el computador posibilita los procesos de conectividad, de comunicaciones y de cálculo de símbolos pertenecientes a un alfabeto.

### 1.5.2 Concepto de Computador

Es una máquina de procesamiento electrónico de datos, que permite recibir datos de entrada, procesar los datos recibidos con acciones aritméticas y lógicas para dar como resultado una información útil al usuario.

El procesamiento de los datos requiere una lógica de programación, la cual se basa en estructuras lógicas que se representan mediante los algoritmos. Estos algoritmos se codifican en un lenguaje de programación que genera un programa (Código C, Java), y los programas se corren en un computador con el fin de obtener la respuesta requerida por el usuario.

El término "computador", "computadora" u "ordenador" se asocia también a las siguientes definiciones:

> Una computadora es un dispositivo electrónico, utilizado para procesar información y obtener resultados, capaz de ejecutar cálculos y tomar decisiones a velocidades millones o cientos de millones más rápidas que puedan hacerlo los seres humanos. (Joyanes, 2008, p. 4).

> Una computadora es una máquina y, como otras máquinas, como un automóvil o una podadora, debe encenderse y luego conducirse, o controlarse, para hacer la tarea que se pretende realizar. En un automóvil, por ejemplo, el control es proporcionado por el conductor, quien se sienta en su interior y lo dirige. En una computadora, el conductor es un conjunto de instrucciones llamado programa. (Bronson, 2007, p. 2).

Es importante destacar en tales definiciones las siguientes categorías conceptuales con relación al término "computador": *i*) Es una máquina o dispositivo, requiere un control de encendido, operación y apagado. *ii*) Necesita un control; a la computadora la controla un programa. *iii*) El programa de computadora permite el procesamiento de lo requerido por el usuario. *iv*) El programa recibe datos, los procesa con operadores aritmético y lógicos, y da la respuesta al usuario convertida en información;

que a su vez, al entrar nuevamente al computador, se vuelve a procesar para generar nuevas informaciones. *v*) Con la información se pueden tomar decisiones; en tal sentido, la información procesada por el computador debe ser útil al usuario. vi) El procesamiento requiere de un lógica de programación. *vii*) La lógica de programación se basa en estructuras mentales o lógicas que sirven para diseñar el flujo de control en el programa. *viii*) Las estructuras lógicas que dan control al programa conforman algoritmos. *ix*) Los algoritmos son entonces conjuntos de reglas que realizan cálculos; y *x*) Los algoritmos requieren de un lenguaje de programación para ser codificados, y así convertirse en un programa de computadora.

El computador, por ser una máquina, requiere de un *hardware*, y para su funcionamiento requiere de un *software*. El hardware son las componentes metálicas del computador y el software es el lenguaje de programación que hace que el computador funcione. El computador funcionando en el interior de un sistema computacional cuando hace procesos de conectividad y comunicaciones necesita el *comware*. Este es una combinación de elementos que permiten al computador conectase y comunicarse formando una red, con el fin de transmitir información entre un computador fuente (o el generador de la información) y un computador destino (o el receptor de la información), como es el caso de la red Internet.

### 1.5.3 Arquitectura de un computador

La computadora como máquina o dispositivo con base en sus funciones de entrada (E), proceso (P) y Salida (S) se compone de tres partes:

- Las unidades de entrada (U/E), tales como el teclado, el ratón (*mouse*) o la pantalla táctil.
- La unidad central de procesamiento (UCP o CPU (*Central Processing Unit*)).
- Las unidades de salida (U/S), como el caso de la pantalla del computador o la impresora.

La CPU (abreviación de *Central Processing Unit*) es la unidad que desempeña la función primaria de un sistema computacional; está compuesta por la unidad de control, la memoria y la unidad aritmético/lógica.

La unidad de control es la parte del computador digital que realiza la secuencia de operaciones e interrupciones de las instrucciones de un programa codificadas en un lenguaje de programación; y a su vez envía las señales apropiadas a los circuitos del computador con el fin de realizar las operaciones aritméticas o lógicas necesarias en la ejecución de las instrucciones del programa almacenado en la memoria del computador.

La memoria es el dispositivo de almacenamiento utilizado por el computador digital con el fin de albergar en ella los datos de entrada, los programas y los resultados. La memoria tiene dos características principales: las direcciones de memoria

y el contenido que se almacena en las direcciones de memoria. Luego, similar a una distribución urbanística de una ciudad: en la dirección Calle 96 n.° 51-32 se encuentra viviendo José Jesús Estrella, siendo la persona en mención el contenido de identidad digital de la vivienda en mención.

Se debe tener en cuenta que existen dos tipos de memoria: la memoria de acceso random y la memoria de solo lectura. La primera, identificada como RAM por sus siglas en inglés (***R****andom* ***A****ccess* ***M****emory*), es una memoria volátil o temporal que contiene el computador y se llama "memoria principal", cuya función es almacenar los datos que se deben procesar, el programa del usuario y los resultados de información obtenidos por el computador para el usuario; los contenidos de la memoria RAM son volátiles, por cuanto se pierden al apagar el computador. La segunda se identifica como ROM, que significa en inglés ***R****ead* ***O****nly* ***M****emory*, o memoria de solo lectura, la cual tiene como principal función almacenar los datos, los programas y los resultados del usuario de una forma permanente, o lo que es equivalente: cuando el computador se apaga las informaciones almacenadas en la ROM no se pierden, siendo, por lo tanto, su almacenamiento permanente.

La unidad aritmético/lógica (U A/L) es la parte del computador que se encarga de hacer las operaciones aritméticas y lógicas requeridas por las instrucciones del programa almacenado en el computador, o lo que es equivalente: "... es la sección de "manufactura" de la computadora" (Deitel & Deitel, 2004, p. 5). La U A/L tiene como función el tratamiento matemático de la lógica formal del programa mediante el uso de un conjunto de símbolos que representan cantidades y relaciones, que en últimas se ejecutan a través de circuitos lógicos que se encargan de hacer las operaciones lógicas propias de la Lógica booleana, tales como la operación AND, OR o la operación NOT, entre otras. Estas operaciones en sus circuitos digitales posibilitan la utilización de dígitos binarios con el fin de ejecutar las instrucciones contenidas en el programa del usuario, y de esta forma resolver a través del programa los problemas de este factibles de tratar computacionalmente a través de una máquina de procesamiento electrónico de datos.

Esquemáticamente, la arquitectura de un computador digital se representa en la figura 1.1.

Con base en la figura 1.1 suponga que el usuario va a realizar la operación matemática $C = A + B$. Esta ecuación matemática para poder ser realizada en una lógica de control algorítmica[3] necesita tres primitivas de control: 1) Leer $a, b$, lleva al computador las variables de entrada $A$ y $B$. 2) Asignar el resultado a la variable $C$, que en notación algorítmica es $c \leftarrow a + b$ (lo cual significa que a la variable $C$ se está llevando el cálculo matemático de la suma de $A+B$). 3) Imprimir $c$, variable que contiene el resultado para el usuario. Una vez se construyen las tres primitivas de control, que forman el algoritmo para sumar dos números, el algoritmo se codifica en un lenguaje de programación (C, C++, Visual C++, Java, ...), y se tiene el programa de sumar dos números en un computador digital. Cuando el programa se

[3]Las estructuras de la lógica de control algorítmica se desarrollan en el capítulo 2 de este texto, con el nombre de "Estructuras básicas para la construcción de algoritmos".

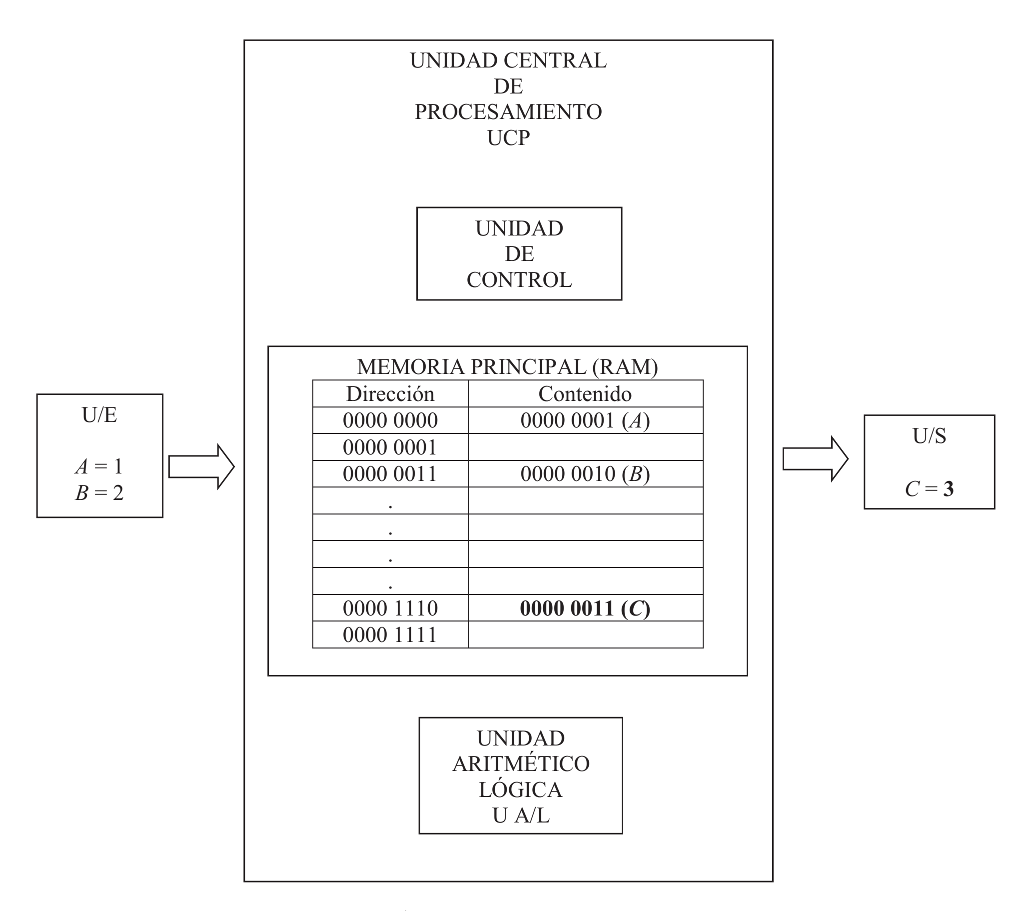



*Figura 1.1* Arquitectura de un computador

corre en el computador, se ejecuta de la siguiente forma: 1) Se ejecuta primero la lógica de control de lectura; o sea, lee las variables $A$ y $B$; suponga que los valores de las variables sean $A = (1)_{10}$ en base decimal y $B = (2)_{10}$, también en base diez; entonces el computador convierte el contenido de las variables dadas en base decimal (10) por el usuario a contenidos en base binaria o base dos (2); luego $A = (1)_{10}$ es igual a $(0001)_2$ y $B = (2)_{10}$ es igual al valor $(0010)_2$ en base binaria. Se debe tener en cuenta que los datos se almacenan en las direcciones de memoria del computador; para el caso de la figura 1.1, la variable $A =$ está almacenada en la dirección binaria de la memoria igual a 0000 y la variable $B =$ está contenida en la dirección de memoria igual a 0011.

Teniendo en cuenta los datos contenidos en las variables almacenadas en mención, la unidad aritmético/lógica realiza los cálculos matemáticos; o sea, suma en binario los valores $0001 + 0010 = 0011$ en base dos y asigna el resultado a la variable $C$; se debe tener en cuenta que la variable $C$, que contiene el resultado del usuario, se encuentra en la dirección de memoria 1110. Finalmente, la primitiva de control Imprima $C$ realiza para el usuario la operación de convertir el contenido de la variable $C = (0011)_2$ de base dos, lo cual es igual a $(3)_{10}$ en base diez, que es el resultado o información útil al usuario que da como resultado el programa de computador ejecutado en el computador; o lo que es equivalente, la unidad de salida, consola del computador, dará como resultado el número decimal **3**, que ya es **información**.

Teniendo en cuenta que para producir la información el computador tiene que transformar el código fuente o el programa escrito en un lenguaje de programación, que es la instrucción en lenguaje C igual a $c = a + b$, instrucción que se transforma a lenguaje objeto o lenguaje binario $(0011 = 0001 + 0010)$ en el computador, se hacen necesarios los conceptos de almacenamiento de la información en el computador y codificación de la información, lo cual se desarrolla a continuación.

### 1.5.4 Almacenamiento de datos en el computador

La unidad básica de almacenamiento de los datos para producir información en un computador es el *bit*. El término "bit" es la abreviatura de "dígito binario" (en inglés ***b**inary **d**igit*) y es una unidad de información correspondiente a una decisión binaria. La decisión binaria tiene dos estados: el estado uno, o sea, el bit es igual a 1, en cuyo caso se asocia con la decisión de SI, o lo que es equivalente, el encendido en un sistema de control de un dispositivo. El estado dos de la decisión mencionada es igual a cero (0), a cuyo estado se le asocia la decisión NO, lo cual significa que el estado del dispositivo es apagado si el bit está referido a su sistema de control de encendido o apagado.

La unión de una secuencia de ocho bits recibe el nombre de "byte". Luego el byte es un término que indica una secuencia de bits consecutivos tratados como una única entidad de información. La unión de varios bytes (cada uno de ellos con secuencias de 8 bits) recibe el nombre de "palabras en código binario". Luego una

palabra de 2 bytes es igual a 16 bits, que representados en cuartetos de bits para una palabra de 4 bytes se presenta en la figura 1.2.

$b_7$ $b_0$

$\text{Palabra}_{4\ \text{bytes}} = 1001\ 0010_{\text{byte1}}\ 1100\ 0011_{\text{byte2}}\ 1001\ 0010_{\text{byte3}}\ 1100\ 0011_{\text{byte4}}$

Donde $b_7$ es el octavo bit del primer byte contado a partir del bit cero $b_0$ de derecha a izquierda (←)

*Figura 1.2* Palabras, bytes y bits para almacenar la información en un computador

Con base en las figuras 1.1 y 1.2, la memoria principal (RAM) de un computador es un conjunto de celdas organizadas de una forma contigua, cada una de las cuales tiene una sola dirección de memoria que identifica la celda, y en la cual se va a almacenar la información. Luego, teniendo en cuenta lo anterior, la capacidad de memoria de un computador puede contener millones de bytes. Estas capacidades de almacenamiento se miden utilizando los múltiplos del byte, que sin hacer una enumeración exhaustiva son kilobyte (1 KB = 1024 bytes); megabyte (1 MB = 1024KB), gigabyte (1 GB = 1024 MB) o terabytes (1 TB = 1024 GB).

La información del usuario que se va a almacenar en el computador puede pertenecer a los dominios cognitivos de: la voz o el tacto (lectura/escritura), la visión (imágenes), el audio (sonido), el olfato (tipos de olor); cada uno de los cuales tiene su respectiva representación en el computador.

Para generar la información útil al usuario, el dato se debe almacenar y procesar en el computador; luego, en síntesis, el objetivo fundamental de todos los programas (generados a partir de la construcción de un algoritmo) es almacenar y procesar datos para generar informaciones como resultados para el usuario. Los tipos de datos según el dominio pueden ser: numéricos (dominio del tacto), alfabéticos (dominio de la voz), de audio (dominio auditivo) o de video (dominio de la visión).

La tabla 1.1 especifica los datos más relevantes para el lenguaje C, indicando para los tipos de datos: sus capacidades de almacenamiento asociadas al tipo de datos definido, el intervalo de valores que puede tener el tipo de dato en el lenguaje mencionado, su palabra clave de identidad en el lenguaje y, finalmente, la declaración de variables para la codificación de las variables del algoritmo al pasarlo al lenguaje de programación C.

Los datos numéricos, alfabéticos o de otro tipo en su significado son realmente átomos unitarios o caracteres que pertenecen a otros tipos sistemas. En el caso de los datos numéricos, se habla de sistemas numéricos binarios $\{1, 0\}$, decimales {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} o hexadecimales {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, $A$, $B$, $C$, $D$, $E$, $F$}; es decir, el sistema binario tiene dos caracteres, 0 y 1. En el caso de datos alfabéticos, se habla del alfabeto latino {A, B, C,. . ., Z, a, b, c,. . ., z}. Luego, cuando un carácter de un sistema numérico o alfabético se introduce en un dispositivo (U/E o U/S) del computador, los caracteres se codifican.

*Tabla 1.1* Tipos de datos

| Dominio Cognitivo | Tipo de dato | Lenguje C | Ejemplos | Almacenamiento | Intervalo | Declaración de variables en el lenguaje C |
|---|---|---|---|---|---|---|
| Lectura/Escritura Números enteros | Entero Cortos | short int | 9, −5 | 2 bytes | [−32768;32767] | short int suma; |
| Lectura/Escritura Números enteros | Enteros | int | +98752, −999936 | 4 bytes | [−2147483648; 2147483647] | int promedio; |
| Lectura/Escritura Números Reales | Punto flotante o Real | float | +777.77, −0.47 | 4 bytes | [1.4013E-45; 3.4028E+38] | float gradiente; |
| Lectura/Escritura Números en Notación científica | Real doble precisión | double long double | 9.4665E+307 | 8 bytes | [4.9406E-324; 1.7977E+308] | long double vel_estelar; |
| Lectura/Escritura Letras del alfabeto | Carácter | char | 'Z', 'n' | 1 byte | 256 caracteres | char palabra_clave; |
| Lectura/Escritura Datos Booleanos | Lógico | Bool | 1 o 0 | 1 byte | Falso = 0; Verdadero = Valor positivo | bool switch_control; |

Fuente: adaptado de Bronson (2007, pp. 2-85).

La codificación de un carácter es la combinación de bits, a través de los cuales se representa dicho carácter. El código de codificación más utilizado es el Código Estándar Americano para el Intercambio de Información, conocido como ASCII por sus siglas en inglés: *American Standard Code for Information Interchange*. En tal sentido, el carácter numérico $(7)_{10}$ en base 10 se representa como $(00000111)_2$ en binario, lo cual corresponde a su código ASCII $(7)_{15}$ en hexadecimal; y la letra mayúscula A corresponde al código ASCII $(65)_{10}$, con su equivalente $(41)_{15}$ en hexadecimal; por lo tanto, la letra A, como carácter, se almacena en el computador como un byte y su contenido es $(01000001)_2$. Los caracteres ASCII con sus equivalentes en binario y hexadecimal se muestran en la tabla 1.2.

Teniendo en cuenta que los datos procesados por el computador son digitales, justificado por el hecho de que su contenido se construye a partir de dígitos binarios, para el correcto entendimiento de los datos que entran, se procesan y se convierten en resultados de información para el usuario es necesario el estudio de los sistemas de numeración, en el marco de la Teoría de la Codificación de la Información.

### 1.5.5 Sistemas de numeración

La justificación por la cual son necesarios los sistemas de numeración al procesar datos en el computador estriba en que los datos solicitados como entrada en los programas del usuario, y los caracteres numéricos y alfabéticos que se procesan en los programas, son el código fuente del programa. Este código es pasado por el computador a código objeto o en base dos, que es, en últimas, es el lenguaje que maneja la máquina. A su vez, la representación alfabética de los datos es única; pero la representación numérica no lo es; de hecho hay enteros cortos (*short int*), hay enteros (*int*) y hay también reales (*float*); pero a su vez, un número real tiene varias representaciones; por ejemplo, puede expresarse en notación científica. Así, el número 777.48459 es equivalente al número $7.7748459 \times 10^2$ (notación científica) o al número $777484.59 \times 10^{-3}$.

Los sistemas de numeración binario, octal, decimal y hexadecimal son los más utilizados para la representación de datos en el computador para pasar el lenguaje fuente a lenguaje objeto o base dos. Estos serán desarrollados en esta sección.

Sistema binario: el alfabeto binario está compuesto por dos cifras $\{0, 1\}$. Por lo tanto, un número binario corresponde a una cadena de bits, la cual expresada como $(a_n a_{n-1} a_{n-2} \ldots a_0)$ forma un sistema numérico posicional, desde la posición 0 hasta la $n$-ésima posición de la cadena. Por ejemplo, en la cadena de bits $C = (1010)_2$, su sistema posicional es de cuatro bits, identificados así: $a_{n=3} = 1$; $a_2 = 0$; $a_1 = 1$ y $a_0 = 0$.

La importancia del sistema posicional consiste en que el número $C$ se puede representar mediante la siguiente ecuación:

*Tabla 1.2* Tabla de caracteres ASCII[4]

| ASCII | Hex | Símbolo | ASCII | Hex | Símbolo | ASCII | Hex | Símbolo | ASCII | Hex | Símbolo |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | NUL | 32 | 20 | (espacio) | 64 | 40 | @ | 96 | 60 | ‘ |
| 1 | 1 | SOH | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | STX | 34 | 22 | ” | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | ETX | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | EOT | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | ENQ | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | ACK | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | BEL | 39 | 27 | ’ | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | BS | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | TAB | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | LF | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | VT | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | FF | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | CR | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | SO | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | SI | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | DLE | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | DC1 | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | DC2 | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | DC3 | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | DC4 | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | NAK | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | SYN | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | ETB | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | CAN | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | EM | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | SUB | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | ESC | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | FS | 60 | 3C | ¡ | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | GS | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | RS | 62 | 3E | ¿ | 94 | 5E | ˆ | 126 | 7E | ˜ |
| 31 | 1F | US | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | □ |

[4]Fuente original:
http://www.sitiosargentina.com.ar/categorias/internet/formatos/ASCII.htm

$$C = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \cdots + a_1 2^1 + a_0 2^0 = \sum_{i=0}^{n-1} a_i 2^i$$

Teniendo en cuenta esta ecuación, la cadena de dígitos binaria $C = (1010)_2$ tiene cuatro cifras binarias; luego a partir de cero, entonces $n = 3$, de derecha a izquierda; entonces el número binario dado es igual a la cantidad $C = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 2 = (9)_{10}$ en base diez. Se debe tener en cuenta que los bits con valor cero no contribuyen en este caso con el posicionamiento de las potencias de base binaria $2^n, 2^{n-1}, 2^{n-2}, \ldots, 2^i, \ldots, 2^3, 2^2, 2^1, 2^0$.

Con base en lo anterior, si se define la base $b$ de un sistema de numeración y $n$ el número de cifras de un número expresado en la base dada, entonces el número $C$ está dado por la siguiente expresión matemática:

$$C = (a_{n-1} \cdots a_1 a_0)_b = \sum_{i=0}^{n-1} a_i b^i$$

donde cada uno de los $a_i$ son las cifras del número $C$ y puede ser una de las cifras $b_0$, $b_1$, ..., $b_{n-1}$ que corresponden a la base $b$.

Se debe tener en cuenta que los sistemas de numeración son bases para la codificación de los datos, y con base en los datos representados en dicha codificación se pueden hacer operaciones. En tal sentido, si se tienen dos operandos binarios, identificados como $b_1$ y $b_2$, si se realiza la operación binaria de la suma de números binarios, entonces la realización de la operación se muestra en la tabla 1.3.

*Tabla 1.3* Operación binaria de suma de bits

| Operando uno $b_1$ | Operador + | Operando dos $b_2$ | Resultado | Acarreo |
|---|---|---|---|---|
| 0 | + | 0 | 0 | |
| 0 | + | 1 | 1 | |
| 1 | + | 0 | 1 | |
| 1 | + | 1 | 0 | 1 |

Sistema octal: los dígitos que componen el alfabeto octal es el conjunto {0, 1, 2, 3, 4, 5, 6, 7}; de este modo, el número $(17)_8$ en base octal es igual a $C = 1 \times 8^1 + 7 \times 8^0 = 8 + 7 = (15)_{10}$.

Sistema decimal: el alfabeto que forma el sistema digital o en base 10 está compuesto por el conjunto de dígitos {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}; utilizando el sistema de posicionamiento decimal de derecha a izquierda sin hacer un enumeración exhaustiva de posiciones, un número está compuesto por unidades ($1 = 10^0$), decenas ($10 = 10^1$), centenas ($100 = 10^2$), unidades de mil ($1000 = 10^3$), decenas de mil ($10000 = 10^4$), centenas de mil ($100000 = 10^5$), unidades de millón ($1000000 = 10^6$), y así sucesivamente. La justificación de la cantidad del número $(9874)_{10}$ con base en el sistema de posicionamiento decimal es $9874 = 9 \times 10^3 + 8 \times 10^2 + 7 \times 10^1 + 4 \times 10^0 = 9 \times 1000 + 8 \times 100 + 7 \times 10 + 4 \times 1 = 9000 + 800 + 70 + 4 = 9874$.

Sistema hexadecimal: por su parte, el sistema de números hexadecimal en cuanto a su alfabeto está compuesto por el conjunto de número y letras {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}, donde las letras en su equivalencia numérica son $A = 10$, $B = 11$, $C = 12$, $D = 13$, $E = 14$, $F = 15$. En este sentido, la cantidad $(FFFF)_{16}$ en base hexadecimal es igual a $15\times 16^3+15\times 16^2+15\times 16^1+15\times 16^0 = 15 \times 4096 + 15 \times 256 + 15 \times 16 + 15 \times 1 = 61440 + 3840 + 240 + 15 = (65535)_{10}$.

---

Ejemplo 1.2 Conversión de bases

---

a) Convertir $(1111)_2$ a base decimal

$(1111)_2 = 1\times 2^3+1\times 2^2+1\times 2^1+1\times 2^0 = 1\times 8+1\times 4+1\times 2+1\times 1 = (15)_{10}$. Lo cual corresponde al sistema de posicionamiento binario 8 4 2 1, con todos los bits de posicionamiento en 1.

b) Convertir $(1010)_2$ a base octal

$$\begin{aligned}(1010)_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0\\ &= 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 0 = (10)_{10}\end{aligned}$$

$(10)_{10} = 10/8 = 1$ con residuo 2; para lo cual se toma el cociente 1 y el residuo 2; entonces $(10)_{10} = (12)_8$.

c) Realizar la operación $(1001)_2+(1210)_3+(1001)_8+(100A)_{16}$, efectuando la suma en decimal:

$$\begin{aligned}(1001)_2 &= 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0\\ &= 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = (9)_{10}.\\ (1210)_3 &= 1 \times 3^3 + 2 \times 3^2 + 1 \times 3^1 + 0 \times 3^0\\ &= 1 \times 27 + 2 \times 9 + 1 \times 3 + 0 \times 1 = (48)_{10}.\\ (1001)_8 &= 1 \times 8^3 + 0 \times 8^2 + 0 \times 8^1 + 1 \times 8^0\\ &= 1 \times 512 + 0 \times 64 + 0 \times 8 + 1 \times 1 = (513)_{10}.\\ (100A)_{16} &= 1 \times 16^3 + 0 \times 16^2 + 0 \times 16^1 + 10 \times 16^0\\ &= 1 \times 4096 + 0 \times 256 + 0 \times 16 + 16 \times 1 = (4112)_{10}.\end{aligned}$$

Luego $(9)_{10} + (48)_{10} + (513)_{10} + (4112)_{10} = (4682)_{10}$.

d) Realizar $(10)_{20}+(9)_{10}+(14)_8+(100)_2$, efectuando la suma en el sistema binario:

$$(10)_{20} = 1 \times 20^1 + 0 \times 20^0 = 1 \times 20 + 0 \times 1 = (20)_{10} = (10100)_2.$$

$(9)_{10} = 9/2 = 4$ (residuo 1), $4/2 = 2$ (residuo 0), $2/2 = 1$ (residuo es 0) $= (1001)_2$, de las divisiones sucesivas se toma el último cociente y los residuos en orden inverso.

$$(14)_8 = 1 \times 8^1 + 4 \times 8^0 = 1 \times 8 + 4 \times 1 = (12)_{10} = (1100)_2.$$

Luego, al realizar la suma en binario:

$$
\begin{aligned}
& & 11\,(\leftarrow)\,\text{Valor del acarreo}\\
(10)_{20} &=& (10100)_2\\
(9)_{10} &=& (01001)_2\\
(14)_8 &=& (01100)_2\\
(100)_2 &=& (00100)_2\\
\text{Resultado} &=& \underline{(101101)_2}\\
&=& (20)_{10} + (9)_{10} + (12)_{10} + (4)_{10}\\
&=& 1\times 32 + 0\times 16 + 1\times 8 + 1\times 4 + 0\times 2 + 1\times 1\\
&=& (45)_{10}.
\end{aligned}
$$

---

Suponga que un computador tiene $x$_Gb (2 gigabytes) de memoria real, una unidad de almacenamiento secundaria de $y$_Tb (1 terabyte) y $n$ ($n = 4$ unidades portables de memoria, cada uno de ellas con $z$_Mb (1000 megabytes) de espacio de almacenamiento. Calcule la capacidad total la máquina en términos de bytes.

El espacio de cada unidad portable es igual a $z$_Mb, o sea, $z \times 1024$ Kb $\Rightarrow z \times 1024 \times 1024$; por las $n$ unidades de almacenamiento, el resultado es $n \times z \times 1024 \times 1024$ bytes.

El espacio de almacenamiento secundario es $y$_Tb $= y \times 1024\,\text{Gb} = y \times 1024 \times 1024\,\text{Mb} = y \times 1024 \times 1024 \times 1024\,\text{Kb} = y \times 1024 \times 1024 \times 1024 \times 1024$ bytes. El espacio de almacenamiento en memoria real es $X$_Gb $= x \times 1024\,\text{Mb} = x \times 1024 \times 1024\,\text{Kb} = x \times 1024\,\text{Mb} = x \times 1024 \times 1024 \times 1024$ bytes.

Luego el espacio total de almacenamiento del computador es igual a

$$(y \times 1024^4 + x \times 1024^3 + n \times z \times 1024^2)\,\text{bytes}$$

En términos de datos concretos, los resultados de las operaciones son:

$$(1 \times 1024^4 + 2 \times 1024^3 + 4 \times 1000 \times 1024^2)\,\text{bytes}$$

$$1 \times 1099511627776 + 2 \times 1073741824 + 4000 \times 1048576$$

$$1099511627776 + 2147483648 + 4194304000 = 1105853415424\,\text{bytes}.$$

## 1.6 Lógica de programación en Ingeniería

La lógica de programación para ingenieros es una herramienta que combina los campos de la lógica humana, la algoritmia y la Ciencia de la Computación en un

sentido Informático, para consolidar las capacidades del ingeniero en identificar, formular y resolver problemas en Ingeniería.

El Consejo de Acreditación para la Ingeniería y la Tecnología *Accreditation Board for Engineering and Technology* (ABET), de los Estados Unidos de América (Wright, 1994, p. 25), la define como

> La profesión en la cual el conocimiento de las ciencias matemáticas y naturales adquirido mediante el estudio, la experiencia y la práctica, se aplica con buen juicio a fin de desarrollar las formas en que se pueden utilizar, de manera económica, los materiales y las fuerzas de la naturaleza en beneficio de la humanidad.

La relación que existe entre la definición de Ingeniería dada por ABET y la lógica de programación es directa, justificada por el siguiente análisis:

La lógica humana, desde el mundo antiguo (los sumeros, los egipcios, los griegos, los romanos) hasta el mundo moderno, ha estado al servicio de la humanidad en obras asociadas a las diferentes ramas de la Ingeniería. Es así como las pirámides egipcias, el panteón como templo romano, o el Faro de Alejandría, construido por los griegos son obras de Ingeniería presentes en la edad actual, para las cuales las diferentes civilizaciones utilizaron el conocimiento matemático a fin de realizar las estructuras arquitectónicas con base en la interrelación matemático-geométrica, dentro de las cuales fueron bases de cálculo los sistemas numéricos y, consecuentemente, las reglas algorítmicas como conjunto organizado de pasos para realizar un proceso o proyecto determinado. La lógica humana para la realización de obras en cualquier área de la Ingeniería se soporta en las Matemáticas y las Ciencias Físicas y Químicas como fundamentos necesarios para entender los fenómenos de la naturaleza y, consecuentemente, en el mundo real, poder identificar y formular problemas en Ingeniería. La lógica humana tiene en la lógica algorítmica una herramienta que permite “en parte” automatizar la lógica humana; es así como cálculos de grandes cantidades en cualquier proyecto de Ingeniería son realizados más eficientemente y con mayor grado de precisión mediante el computador, para cuyo funcionamiento son necesarios los algoritmos, como paso previo a la codificación y, en últimas, a la obtención de un programa para funcionar en un computador digital. El paso de la lógica humana a la lógica algorítmica, y de esta a la lógica de programación en las fases de resolución de un problema en Ingeniería, requiere las mismas etapas de la definición de Ingeniería: el estudio, la experiencia y la práctica. Lo anterior significa que es necesario el estudio de las estructuras lógicas básicas para la construcción de algoritmos, la experiencia acumulada en la construcción de algoritmos y, finalmente, la práctica de la lógica de programación para obtener soluciones en lógicas programables aceptables y representativas de la práctica en Ingeniería en sus condiciones de utilidad al usuario.

Cualquiera de las lógicas en análisis (humana, algorítmica o de programación) no es útil si no es aplicable con buen juicio. Luego, el estudio, la experiencia y la práctica en el marco de la lógica de programación debe ser aplicado según el sistema de razonamiento lógico del ingeniero, o lo que es equivalente, con una condición de

verdad validada por el constructor del algoritmo o del programa de computador focalizado a beneficiar al usuario, tanto en el procesamiento de los datos como en las informaciones derivadas del programa de computador.

Con base en lo anterior, en el marco de una sociedad globalizada y basado en la generación de información y tiendo en cuenta el paradigma socio-técnico, se acepta el hecho que

> ... el uso de las máquinas computacionales ha llegado a ser amplio, y hay una necesidad entendida del alcance e impacto de lo que se llama la Revolución de la Información o la Edad de la Información Digital. (Committee for the Workshops on Computational Thinking & National Research Council, 2010, p. 7)

Luego, el trabajo del ingeniero en la sociedad del conocimiento, soportado por la sociedad de la información, requiere para el profesional como mínimo interactuar necesariamente con las máquinas generadoras de información digital (computador). Ahora bien, para construir proyectos ingenieriles soportados en computadores es necesario que el ingeniero conozca la lógica de programación; y la lógica de programación se basa necesariamente en la lógica algorítmica, porque dicha lógica se genera con base en la lógica humana. Las acciones de Ingeniería de teoría, abstracción y diseño en su relación con el computador dependen de la lógica de programación para beneficiar a los usuarios de cualquier profesión de la Ingeniería con la información. Así, las bases de datos de información, que constituyen el núcleo teórico de conocimientos en cualquier profesión humana, no hubieran sido posibles sin procesos de lógica algorítmica y de programación; porque se debe estructurar la base de datos, poblarla con datos primarios y mantenerla. La acción de abstracción en cualquier obra de Ingeniería requiere lógica; y la lógica de programación con base en la algoritmia facilita la abstracción de modelos representativos de fenómenos del mundo real; de hecho, casos como la construcción de un transbordador espacial, en el cual confluyen varias profesiones de la Ingeniería, no se realiza en su fase experimental sino a través de simulaciones de su construcción física; y los simuladores computacionales, al permitir abstraer condiciones de fenómenos reales, se soportan en la lógica de programación. El diseño de Ingeniería con base en la lógica de programación posibilita el avance de áreas tales como CAD/CAM/CAE (*Computer Aided Desing/Computer Aided Manufacturing/Computer Aided Enginnering*, respectivamente), debido a que

> CAE significa Ingeniería Asistida por Computador y primariamente hace funcionar dos áreas: la Ingeniería de Software y la tecnología. Esta última, la manufactura la usa en conjunto con el Diseño Asistido por Computador (CAD) para hacer la ingeniería, el análisis y la optimización del diseño de producto. Diseño, análisis e Ingeniería basada en conocimiento (KBE[5]) que son aplicaciones que ayudan a los manufactureros en los conceptos de diseño a través de la simulación del comportamiento de la física de un producto en su ambiente de operación e incluyen el conocimiento y experiencia del diseñador en el proceso de manufacturación. (Mirman & McGill, 2004, p. 159)

[5]KBE: Knowledge-based engineering, o Ingeniería Basada en Conocimiento.

Con base en lo anterior, es necesario en la formación de ingenieros incluir los conceptos relacionados con la construcción de algoritmos como herramientas que unidas a la lógica humana y al conocimiento experto asociado a cada profesión hacen posibles la construcción, solución y desarrollo de sistemas de información, que construidos en software con base en la algoritmia y funcionando sobre dispositivos computacionales permiten aplicar los conocimiento de la matemáticas, la ciencia y la Ingeniería en el desarrollo de problemas neurálgicos para el desarrollo de la humanidad y para la conservación del planeta.

Por lo tanto, reconociendo la importancia de la lógica algorítmica en el desarrollo de la lógica humana y enfatizando la necesidad del desarrollo de la lógica de programación en función de la lógica algorítmica, y dada la dinámica de los desarrollos en Ingeniería en la sociedad del conocimiento, soportados por la Ciencia de la Computación, se hace necesario el entendimiento de la lógica algorítmica y la formación de ingenieros en Algoritmia y Programación, que sirven de sustento al desarrollo de proyectos informáticos en áreas tan neurálgicas para el desarrollo humano como la salud, el medio ambiente, la generación energética, en la óptica de que los proyectos informáticos desarrollados necesariamente por la dinámica de la ciencia y la técnica en Ingeniería han de ser multidisciplinarios, y una vez desarrollados se debe evaluar el impacto de las soluciones algorítmicas propuestas en el desarrollo de los proyectos en contextos nacionales, mundiales, económicos, ambientales y sociales, he ahí la importancia de la enseñanza y el aprendizaje de la lógica algorítmica para los actuales y futuros ingenieros al servicio de una sociedad información y global basada en el conocimiento humano pero con sentido humanista. Luego, la construcción de algoritmos para el desarrollo de los proyectos mencionados es el contenido central del próximo capítulo.

## 1.7 Conclusiones

Los temas desarrollados en este primer capítulo permiten concluir que:

- El desarrollo y avance de la ciencia y la técnica a través de la humanidad ha estado soportado por la Algoritmia, la Matemática y la Ciencia de la Computación (*Computer Science*).
- La algoritmia, en su función esencial de estudio de la construcción, el desarrollo, funcionamiento y avance de los algoritmos, está directamente correlacionada con la lógica humana; en tal sentido, un desarrollo del pensamiento del sujeto a nivel formal permite la construcción de lógicas formales; y a su vez, el desarrollo de estructuras algorítmicas permite el avance de la lógica formal de la persona. Ello implica la importancia de los algoritmos en el desarrollo presente y futuro de cualquier profesión al servicio de la sociedad.
- Los conceptos bases del funcionamiento de un algoritmo dependen de dos bases: la lógica humana como la lógica algorítmica aplicada solución a un pro-

blema representado en un conjunto de reglas organizadas hacia un fin útil o algoritmo; luego, el correcto conocimiento de las estructuras bases para la construcción de algoritmos soporta el éxito de la condición de verdad del algoritmo, unida simpre al correcto ejercicio de la razón en el cumplimiento de la validez y utilidad del algoritmo, que con relación a la solución de un problema, el algoritmo debe necesariamiente hacer lo que debe hacer, o sea, solucionarle el problema al usuario.

- El cumplimiento de la buena solución de un problema fundamentado en la utilización de las lógicas humana racional y algorítmica son bases para generar, de resultas, la lógica de programación, siendo un programa un algoritmo codificado en un lenguaje de programación entendible para una máquina electrónica de datos-computador.
- Las lógicas humana, algorítmica, de programación en su proceso integrado generan el software (*logiciel*) de los sistemas informáticos; software que necesita de dispositivos de hardware o el computador para su funcionamiento.
- La integración software-hardware se complementa con los procesos de comware o generación de acciones de conectividad y comunicaciones entre varios computadores que funcionan con piezas de software, lo cual conforma la red, en el marco del paradigma socio-técnico.
- La red (Internet) ha permitido la creación de grupos sociales; cada uno de ellos siendo usarios de múltiples piezas de software y hardware interconectadas que son funcionales por la operación de lógicas de programación, y en síntesis, dichas piezas operan con bases en la lógica algorímica.
- La red en el marco del presente y el desarrollo futuro de la Informática y la Ciencia de la Computación tiene un impacto en la política, la economía, la salud y la vida de la actual y futura sociedad del conocimiento predictivo, en el sentido de que el conocimiento es la base de las decisiones y, consecuentemente, el fundamento de los sistemas predictivos en función de informaciones almacenadas en series de tiempo respresentativos de fenómenos de la realidad humana.
- Por lo tanto, el recorrido realizado desde la Algoritmia a la Sociedad del Conocimiento induce el conocimiento fundamental que en la vida moderna ha de tener el sujeto de los conceptos que soportan la construcción de algoritmos; para lo cual los siguientes capítulos de este libro permiten al lector la construcción de sus propios algoritmos y el avance de su pensamiento formal y, consecuentemente, de su nivel inteligencia.

# Bibliografía

Bronson, G. (2007). *C++ Para Ingeniería y Ciencias*. México, D.F: Thomson.

Burgin, M. (2009). *Theory of information: Fundamentality, diversity and unification*. SGP: World Scientific Publishing Co.

Chabert, J.-L. & Barbin, E. (1999). *A history of algorithms: From the pebble to the microchip*. Berlin: Springer.

Committee for the Workshops on Computational Thinking & National Research Council (2010). *Report of a workshop on the scope and nature of computational thinking*. Washington: National Academies Press.

Coutard, O. (1999). *Governance of large technical systems*. Florence: Routledge.

Dean, N. (2003). *Logic and language*. Gordonsville: Palgrave Macmillan.

Deitel, H. & Deitel, P. (2004). *Cómo Programar en Java*. México: Pearson Eduction México, S. A. de C.V.

Hopcroft, J. & Ullman, J. (1969). *Languages and their relation to automata*. USA: Addison-Wesley Publishing Company.

Johansen, B. (1992). *Introducción a la teoría general de sistemas*. México, D.F: Limusa.

Joyanes, L. (2008). *Fundamentos de programación*. Madrid: McGraw-Hill/Interamericana de España, S.A.U.

Mirman, I. & McGill, R. (2004). CAD/CAM/CAE. En Geng, H., editor, *Manufacturing Engineering Handbook*. McGraw-Hill Professional Publishing, New York.

Mishra, R. C. & Sandilya, A. (2009). *Reliability and quality management*. Daryaganj: New Age International.

Myers, M. & Kaposi, A. (2004). *First systems book: Technology and management*. Singapore: Imperial College Press, 2 edition.

Wear, L., Pinkert, J., Wear, L., & Lane, W. (1991). *COMPUTERS An introduction to Hardware and Software Design*. USA: McGraw-Hill, Inc.

Wright, P. (1994). *Introducción a la Ingeniería*. Wilmington: Addison-Wesley Iberoamericana, S. A.

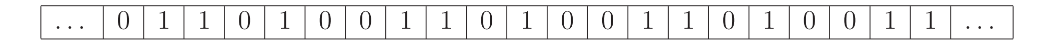
# Capítulo 2

# Datos e información

Los datos constituyen la “materia prima” de la información. Un dato representa un número, una letra o un carácter que puede tener o no significado para quien lo recibe. Si tomamos un conjunto de datos aleatoriamente, es posible que no transmita ningún mensaje coherente o que no haya relación alguna entre sus elementos, es decir, los datos no tienen la capacidad de representar información por sí solos.

La información es la asociación coherente y con significado dentro de un contexto definido de datos individuales o de conjuntos de datos; por lo tanto, la información siempre tendrá significado para quien la recibe.

Para la computadora los datos tendrán significado si están codificados en el mismo lenguaje que estas utilizan. El lenguaje utilizado por las computadoras es el sistema numérico binario (*0 y 1*), representado por impulsos eléctricos (*apagado y encendido*). Por esto, los datos se almacenan en la memoria de la computadora en forma de un arreglo lineal de ceros y unos, como se muestra a continuación:



| | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ... | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | ... |

Cada una de las celdas del arreglo anterior se conoce como bit.

## 2.1 Bit y Byte

### 2.1.1 Bit

Es la contracción o acrónimo de dígito binario (***Bi***nary digi***t***). El bit puede ser considerado como la unidad más pequeña de información. Se utiliza fundamentalmente en la realización de las denominadas operaciones booleanas: AND, OR, NOT y XOR. Sin embargo, se necesita más de un bit para la representación de datos (*números, letras, símbolos*), por lo que se introduce el concepto de **byte**.

### 2.1.2 Byte

Se denomina byte u octeto a un conjunto de ocho bits, en los que se almacenan números binarios de ocho dígitos. El byte es la unidad básica de medida de la capacidad de la memoria, o cualquier dispositivo de almacenamiento de una computadora. En un byte se representa el código ASCII de un carácter . Para almacenar cadenas de caracteres se utilizan conjuntos de mayor tamaño, conocidos como múltiplos del byte, pero antes de definirlos resolveremos el siguiente interrogante.

¿Por qué un byte tiene ocho bits y no siete u once, por ejemplo?

La razón es simple. Cuando se diseñó la computadora era necesario asignar códigos a un conjunto de caracteres, alrededor de ciento cincuenta y cuatro, que fueron clasificados en **numéricos** (0, 1, ..., 9), **alfabéticos** (a, b, c, ..., x, y, z, A, B, C, ..., X, Y, Z), **especiales** ($\sharp, @, |, \%, ..., ", !$) y de **control**, *retorno de carro o enter, avance de línea, imprimir pantalla* y otros. Mediante estos códigos se debe suministrar la información a la máquina a través de un dispositivo de entrada, que generalmente es el teclado. Para generar estos códigos son necesarios los grupos de ocho bits, como se muestra a continuación.

Con un bit solo se pueden generar $2^1 = 2$ códigos distintos:
*(0, 1).*

Con dos bits se pueden generar $2^2 = 4$ códigos distintos:
*(00, 01, 10, 11).*

Con tres bits se pueden generar $2^3 = 8$ códigos distintos:
*(000, 001, 010, 011, 100, 101, 110, 111).*

Si continuamos razonando de esta forma, es fácil darse cuenta de que con siete bits pueden generarse $2^7 = 128$ códigos, que es inferior a los 154 que se necesitaban inicialmente.

Con ocho bits se pueden generar $2^8 = 256$ códigos; con estos era posible asignar códigos a los 154 caracteres, y además quedaban a disposición de los diseñadores 102 códigos para nuevos requerimientos o necesidades; esto fue aprovechado y se constituyó en un estándar que se conoce como códigos ASCII (American Standard Code for Information Interchange) .

Este estándar fue definido para 7 bits (128 códigos) y para 8 bits (256 códigos), los cuales se muestran en la tabla 1.2.

Los 32 primeros caracteres y el último son caracteres de control y no son imprimibles. Los siguientes son los significados de los caracteres de control:

| | | | |
|---|---|---|---|
| NULL | Nulo | DC1 | Control de dispositivo 1 |
| SOH | Comienzo de cabeza | DC2 | Control de dispositivo 2 |
| STX | Comienzo de texto | DC3 | Control de dispositivo 3 |
| ETX | Final de texto | DC4 | Control de dispositivo 4 |

| | | | |
|---|---|---|---|
| EOT | Fin de transmisión | NAK | Acuse de recibo negativo |
| ENQ | Petición, consulta | SYN | Sincronización |
| ACK | Acuse de recibo | ETB | Final del bloque de transmisión |
| BEL | Pitido | CAN | Anulación |
| BS | Retroceso de un espacio | EM | Fin de soporte (de cinta, etc.) |
| HT | Tabulación horizontal | SUB | Sustituir |
| LF | Saltar a la línea siguiente | ESC | Escape |
| VT | Tabulación vertical | FS | Separador de archivo |
| FF | Alimentación de hoja | GS | Separador de grupo |
| CR | Retorno de carro | RS | Separador de registro |
| SO | Fuera de código | US | Separador de sub-registro |
| SI | Dentro de código | DEL | Borrar |
| DLE | Escape de enlace de datos. Carácter de control que cambia el significado del carácter que se le sigue. | | |

A este conjunto de caracteres se le denomina "código ASCII extendido". Los 32 primeros caracteres de control son los mismos del conjunto de caracteres ASCII de 7 bits.

## 2.2 Múltiplos y submúltiplos del byte

**Nibble**
Un nibble es un conjunto de cuatro bits, conocido también como "cuarteto o semiocteto", que permite representar un número binario de cuatro dígitos. Estos números van desde el 0000 ($0_{16}$) al 1111 ($F_{16}$) y corresponden a un carácter del sistema numérico hexadecimal. Los únicos submúltiplos del byte son el nibble y el bit.

**Word, Half Word, Double Word**
El Word (Palabra) , el Half Word (Media Palabra) y el Double Word (Doble Palabra) son múltiplos del byte. Un Word es el número máximo de bits con los que trabaja un procesador de manera simultánea. Existen procesadores de 8, 16, 32 hasta 64 bits, y el diseño de los sistemas operativos está restringido a la capacidad de los procesadores.

El Half Word y el Double Word son, entonces, la mitad y el doble de un Word, respectivamente. La longitud o el número de bits de estos múltiplos varía de acuerdo con el tamaño de la palabra; en este caso definiremos la longitud de una palabra como 32 bits para establecer la similitud con la mayoría de procesadores que se fabrican hoy en día.

Los múltiplos del byte más conocidos por su uso frecuente en la especificación de la capacidad de los dispositivos de almacenamiento son: kilobyte (kB[1]), megabyte (MB), gigabyte (GB) y terabyte (TB). Estos prefijos (*kilo, mega, giga, tera*) están definidos en el Sistema Internacional de Medidas (*SI*).

*Tabla 2.1* Tabla de múltiplos y submúltiplos del byte

| Múltiplos/Submúltiplos | n° de bytes | n° de bits |
|---|---|---|
| Bit ($b$) | $2^{-3}$ | $2^0$ |
| Nibble | $2^{-1}$ | $2^2$ |
| Byte ($B$) | $2^0$ | $2^3$ |
| Half Word / Media Palabra | $2^1$ | $2^4$ |
| Word / Palabra ($P$) | $2^2$ | $2^5$ |
| Double Word / Doble Palabra | $2^3$ | $2^6$ |
| Kilobyte ($KB$) | $2^{10}$ | $2^{13}$ |
| Megabyte ($MB$) | $2^{20}$ | $2^{23}$ |
| Gigabyte ($GB$) | $2^{30}$ | $2^{33}$ |
| Terabyte ($TB$) | $2^{40}$ | $2^{43}$ |
| Petabyte ($PB$) | $2^{50}$ | $2^{53}$ |
| Exabyte ($EB$) | $2^{60}$ | $2^{63}$ |
| Zettabyte ($ZB$) | $2^{70}$ | $2^{73}$ |
| Yottabyte ($YB$) | $2^{80}$ | $2^{83}$ |

Se plantea el siguiente interrogante:

¿POR QUÉ UN KILOBYTE CORRESPONDE A $2^{10} = 1024$ BYTES Y NO A 1000 BYTES, COMO LO SUGIERE EL PREFIJO *kilo*, EN EL SISTEMA INTERNACIONAL DE MEDIDAS?

En el sistema numérico decimal, de base $b = 10$, el prefijo *kilo* es equivalente a $10^3 = 1000$; por ejemplo, 1 kilogramo equivale a 1000 gramos, un kilómetro a 1000 metros. Los diseñadores de las computadoras intentaron darle el mismo significado, sin embargo, la máquina trabaja con el sistema numérico binario, de base $b = 2$.

Así surge la siguiente ecuación: $2^x = 10^3$, donde $x$ debe ser un número entero. Las posibles soluciones fueron $x = 9$ o $x = 10$, con lo que $2^9 = 512$ y $2^{10} = 1024$; la segunda solución, $x = 10$, producía un resultado más cercano a 1000, y fue la escogida. Esta situación es un poco desafortunada, ya que profesionales de otras áreas cuestionan el uso del prefijo *kilo* en este contexto; algunos incluso proponen que se utilice el prefijo *kiwi*.

[1]Tenga en cuenta que en minúsculas 'k' es el símbolo de la unidad adecuada para el prefijo kilo. 'K' mayúscula es propiamente el símbolo de la unidad para la unidad de temperatura termodinámica Kelvin. Sin embargo, es muy común dentro de la industria de la computación binaria al indicar la capacidad, especialmente en la literatura de marketing y envasado del producto, el uso de K mayúscula y sin espacio (1 KB), pero '1 KB' no es incorrecto, pero a menudo se considera más adecuado en lenguaje técnico.

## 2.2.1 Ejemplos

- **¿Cuántos caracteres pueden almacenarse en un área de memoria de 128 kilobytes?**

  Asumiendo 1 byte $\equiv$ 1 carácter, se tiene que

$$128\,\text{KB} = 2^7\,\text{KB} \cdot \frac{2^{10}\,\text{B}}{1\,\text{KB}} = 2^{17}\,\text{B}$$

- **¿Cuántas palabras hay en 48 gigabytes?**

$$48\,\text{GB} = 3 \cdot 2^4\,\text{GB} \cdot \frac{2^{30}\,\text{B}}{1\,\text{GB}} \cdot \frac{1\,\text{P}}{2^5\,\text{B}} = \frac{3 \cdot 2^{34}\,\text{P}}{2^5} = 3 \cdot 2^{29}\,\text{P}$$

- **¿Cuántos bits hay en 12 megapalabras?**

$$12\,\text{MP} = 12\,\text{MP} \cdot \frac{2^{20}\,\text{P}}{1\,\text{MP}} \cdot \frac{2^5\,\text{B}}{1\,\text{P}} = 12 \cdot 2^{25}\,\text{B} = 402\,653\,184\,\text{B} = 48\,\text{MB}$$

- **¿Cuántos bits son necesarios para representar un texto que contiene 458 megabytes?**

$$458\,\text{MB} = 458\,\text{MB} \cdot \frac{2^{23}\,\text{B}}{1\,\text{MB}} = 3\,841\,982\,464\,\text{B}$$

- **¿Cuántos terabytes hay en 0,048 yottabytes?**

  1 terabyte $\equiv 2^{40}$ bytes
  1 yottabyte $\equiv 2^{80}$ bytes

$$0,048\,\text{YB} = 0,048\,\text{YB} \cdot \frac{2^{80}\,\text{B}}{1\,\text{YB}} \cdot \frac{1\,\text{TB}}{2^{40}\,\text{B}} = 0.048 \cdot 2^{40}\,\text{TB}$$

- **¿Cuántos nibbles pueden almacenarse en un dispositivo de almacenamiento extraíble de 0,0001 terabytes?**
  1 terabyte $\equiv 2^{40}$ bytes
  1 nibble $\equiv 2^{-1}$ bytes

$$\begin{aligned} 0,0001\,\text{TB} &= 0,0001\,\text{TB} \cdot \frac{2^{40}\,\text{B}}{1\,\text{TB}} = 109\,951\,162.8\,\text{B} \cdot \frac{1\text{ nibble}}{2^{-1}\,\text{B}} \\ &= 219\,902\,325.6\text{ nibbles} \end{aligned}$$

- **¿Cuántos exabytes hay en 458 697 doblepalabras?**
  1 Exa Byte $\equiv 2^{60}$ bytes
  1 Doble Palabra $\equiv 2^3$ bytes

$$458.697\,\text{DP} = 458.697\,\text{DP} \cdot \frac{2^3\,\text{B}}{1\,\text{DP}} \cdot \frac{1\,\text{EB}}{2^{60}\,\text{B}} = 458\,697 \cdot 2^{-57}\,\text{EB}$$

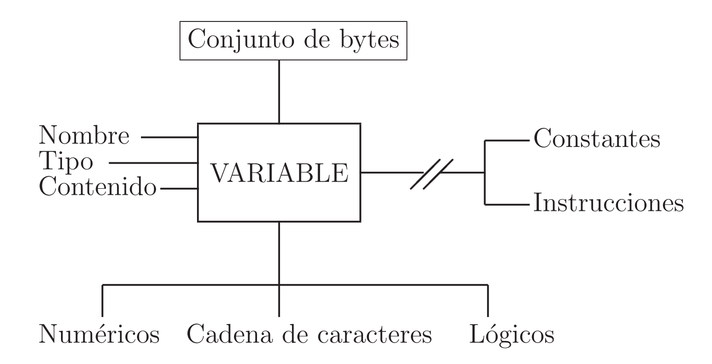
# 2.3 Variables y tipos predefinidos

## 2.3.1 Variables

Una *variable* es un conjunto de bytes en la memoria, referenciado por un nombre, donde se almacena el valor correspondiente a un dato. Dicho valor puede modificarse cuando un programa lo requiera. El nombre de una variable se construye con una o más letras seguidas de un número y/o más letras. No se permiten caracteres especiales (♯, @, |, %, ..., ", !) en los nombres de las variables. Este nombre es elegido por el usuario cumpliendo el criterio anterior. Para que estén correctamente definidas las variables hay que especificar:

- Su nombre.
- El tipo de dato: numérico, carácter, cadena de caracteres o booleano.
- Su valor inicial, el cual es opcional, porque al no inicializar la variable, esta tomará el valor previamente almacenado en el conjunto de bytes que le corresponda.

Cuando una variable se ha declarado de un cierto tipo de dato, solamente puede asignársele datos del mismo tipo. Por ejemplo, una variable que ha sido declarada de tipo alfabético no puede almacenar datos de tipo numérico. Si se intenta asignar a una variable un valor de un tipo de dato que no corresponde con su declaración, se produce un error de tipo.




*Hay otras características de las variables (la dirección y el tamaño) que trataremos posteriormente.*

### 2.3.2 Tipos de datos

#### Datos numéricos

Los *datos numéricos* están contenidos en el conjunto de los números. Existen dos clases de datos numéricos: número entero y número real.

- *Entero*: este tipo de variable admite todos los valores de los números enteros, negativos y positivos. Son los números de valor completo, no se permiten partes fraccionarias o decimales. Se les llama también números de coma fija.

  Por ejemplo: $5, -6, 2000, 3456, -981$.

- *Real*: este tipo de variable admite todos los valores de los números reales, negativos y positivos. Estos números se conocen como compuestos, constan de una parte entera y una parte fraccionaria o decimal. Los números reales pueden representarse en notación científica o en coma flotante.

  Los números reales contienen a los números enteros, por lo tanto, un dato de tipo entero puede ser presentado también como un dato de tipo real con parte decimal nula.

  Por ejemplo: $5.98, -67.21, 0.0201, -3456.00, -981.6788$.

#### Datos carácter y cadena de caracteres

Los datos del tipo *carácter* están contenidos en el conjunto finito de caracteres: alfabéticos, numéricos y especiales. Cuando una variable es declarada de este tipo, admitirá un solo carácter del conjunto de caracteres establecido para la computadora. Por lo tanto, la longitud de una variable tipo carácter siempre será un byte.

Por ejemplo: ‘a’,‘4’,‘G’,‘Z’,‘#’,‘@’.

Los datos del tipo *cadena de caracteres* son sucesiones de caracteres con longitud finita. La longitud de una variable de este tipo es el número de caracteres que la componen.

Por ejemplo: “casa”, “7850”, “GABT”, “Zapato”, “(definición)”, “@jut”, “911”.

#### Datos lógicos

Los datos tipo lógico pueden tomar uno de los dos valores booleanos: VERDADERO (*true*) o FALSO (*false*). Estos dos valores tienen correspondencia con los dígitos del sistema binario: *0* para indicar falso y *1* para indicar verdadero.

### 2.3.3 Tipos Predefinidos

Mediante los lenguajes de alto nivel los programadores pueden definir sus propios tipos de datos, y se conocen como tipos predefinidos. Para los lenguajes de programación *C++* y *Java*, los tipos predefinidos más utilizados son:

- Entero:
    - int (*Variable entera en C++ y Java*)
    - Integer (*Objeto de tipo entero en Java*)
- Real:
    - float (*Variable real de precisión simple en C++ y Java*)
    - double (*Variable real de precisión doble en C++ y Java*)
    - Float (*Objeto de tipo real de precisión simple en Java*)
    - Double (*Objeto de tipo real de precisión doble en Java*)
- Carácter:
    - char (*Variable de tipo carácter en C++ y Java*)
- Cadena de Caracteres:
    - string (*Variable de tipo cadena de caracteres en C++*)
    - String (*Objeto de tipo cadena de caracteres en Java*)
- Lógico o booleano:
    - bool (*Variable de tipo booleano en C++*)
    - boolean (*Variable de tipo booleano en Java*)
    - Boolean (*Objeto de tipo booleano en Java*)
- Valores Predefinidos (*C++ y Java*) :
    - Verdadero: **true**
    - Falso: **false**
    - Valor nulo: **null**

Las variables se representan en grupos de bits de diferente tamaño; pueden ser de 8, 16, 32 y hasta 64 bits. Debido a que los recursos de las máquinas son limitados, se ha establecido para cada tipo de variable un número máximo de bits para representar su valor. Por ejemplo, en la mayoría de las computadoras, una variable de tipo *int* dispone máximo de 16 bits para representar su valor numérico; por ende, en este

caso el máximo número entero representado es $2^{16} - 1 = 65\,535$, y corresponde al número binario que contiene 16 unos: 1111111111111111.

Además, con el objetivo de aprovechar al máximo los recursos disponibles de la máquina se han definido modificadores de longitud para el tamaño de las variables. Existen dos modificadores de longitud: *short* y *long*, los cuales reducen a la mitad o duplican, respectivamente, el número de bits en memoria reservados para la variable. Por ejemplo, si agregamos el modificador *long* a una variable de tipo *int*, tendríamos disponibles 32 bits para representar su valor.

Por ejemplo: *long int*, *short float*, *long double*.

En el sistema numérico binario, para representar números positivos o negativos se utiliza el bit más a la izquierda del número. Si el bit más a la izquierda es *0*, el número es positivo; si es *1*, el número es negativo. Por esto existen dos modificadores adicionales : *signed* (*con signo*) y *unsigned* (*sin signo*). Si queremos representar una variable de tipo *int con signo*, el número mínimo que se puede representar sería -32 768 y el número máximo sería 32 767. Si contamos cuántos números hay en este intervalo, obtendríamos $2^{16} - 1$ números.

*Tabla 2.2* Tipos de Datos (*Cota Mínima y Máxima*)

| Tipo de dato | n° de bits | Valor mínimo | Valor máximo |
|---|---|---|---|
| short int | 8 | $-128$ | 127 |
| long int | 32 | $-2\,147\,483\,647$ | 2 147 483 647 |
| signed int | 16 | $-32\,768$ | 32 767 |
| int (unsigned int) | 16 | 0 | 65 535 |
| float | 32 | $-3.4 \cdot 10^{38}$ | $3.4 \cdot 10^{38}$ |
| double | 64 | $-1.7 \cdot 10^{308}$ | $1.7 \cdot 10^{308}$ |
| long double | 80 | $-3.4 \cdot 10^{4932}$ | $3.4 \cdot 10^{4932}$ |

### 2.3.4 Ejemplos de declaraciones de variables en C++ y Java

- `Entero w`
  **Nombre:** w
  **Tipo:** Entero
  **Contenido:** No asignado
  `int` $w$; *Declaración en C++ y Java*

- `Real saldo_mensual`
  **Nombre:** saldo_mensual
  **Tipo:** Real
  **Contenido:** No asignado
  `float` *saldo_mensual*; *Declaración en C++ y Java, Real de precisión simple*
  `double` *saldo_mensual*; *Declaración en C++ y Java, Real de doble precisión*

- `Entero` $x \longleftarrow 5$

**Nombre:** x
**Tipo:** Entero
**Contenido:** 5
`int` $x = 5$; *Declaración en C++ y Java*

- `Real` $x \longleftarrow 9$
  **Nombre:** x
  **Tipo:** Real
  **Contenido:** 9
  `float` $x = 9$; *Declaración en C++ y Java, Real de precisión simple*
  `double` $x = 9$; *Declaración en C++ y Java, Real de doble precisión*

- `Real` $salario_minimo \longleftarrow 408000$
  **Nombre:** salario_minimo
  **Tipo:** Real
  **Contenido:** 408000
  `float` $salario_minimo = 408000$; *Declaración en C++ y Java, Real de precisión simple*
  `double` $salario_minimo = 408000$; *Declaración en C++ y Java, Real de doble precisión*

- `Cadena` *linea*
  **Nombre:** linea
  **Tipo:** Cadena
  **Contenido:** No asignado
  `string` *linea*; *Declaración en C++, variable string*
  `String` *linea*; *Declaración en Java, objeto String*

- `Cadena` *cad* ⟵ “*cazador*”
  **Nombre:** cad
  **Tipo:** Cadena
  **Contenido:** “*cazador*”
  `string` *cad* = “*cazador*”; *Declaración en C++, variable string*
  `String` *cad* = “*cazador*”; *Declaración en Java, objeto String*

- `Cadena` *info* ⟵ `‘‘Hoy es miércoles 24’’`
  **Nombre:** info
  **Tipo:** Cadena
  **Contenido:** *“Hoy es miércoles 24”*
  `string` *info* = *“Hoy es miércoles 24”*; *Declaración en C++, variable string*
  `String` *info* = *“Hoy es miércoles 24”*; *Declaración en Java, objeto String*

- `Booleano sw`
  **Nombre:** sw
  **Tipo:** Booleano
  **Contenido:** No asignado
  `bool sw;` *Declaración en C++*
  `boolean sw;` *Declaración en Java*

- `Booleano` $sw \longleftarrow false$
  **Nombre:** sw
  **Tipo:** Booleano
  **Contenido:** false
  `bool sw = false;` *Declaración en C++*
  `boolean sw = false;` *Declaración en Java*

## 2.4 Operadores

Todos los símbolos que representan enlaces entre cada uno de los argumentos que intervienen en una operación se llaman operadores, y se utilizan para construir expresiones. Los operadores se clasifican en aritméticos, relacionales y lógicos.

### 2.4.1 Aritméticos

Se utilizan para formar expresiones cuyo resultado será un valor numérico. Junto con las variables de tipo numérico dan lugar a las expresiones aritméticas.

- $+$ Suma
- $-$ Resta, Negación
- $\cdot$ Multiplicación
- $\wedge$ Potenciación
- / División real
- DIV División entera
- MOD Residuo de la división entera

Estos operadores son binarios, es decir, admiten dos operandos: uno a la izquierda y otro a la derecha, y tienen un único resultado. Excepto en el caso del operador "-", es binario cuando indica resta $(X - Y)$ y es unario cuando indica la negación $(-X)$.

Los operadores DIV y MOD no están definidos para todos los lenguajes de programación. En el caso de los lenguajes C++ y Java, estos operadores se representan de la siguiente manera:

- número DIV divisor: `int(numero / divisor)`
- número MOD divisor: `numero % divisor`

### 2.4.2 Relacionales o Condicionales

Se utilizan para formar expresiones booleanas, es decir, expresiones que al ser evaluadas producen un valor booleano : VERDADERO o FALSO.

- $<$ Menor que
- $=$ Igual
- $>$ Mayor que
- $<=$ Menor o igual que
- $>=$ Mayor o igual que
- $\neq$ ó $<>$ Distinto de

Cuando se comparan caracteres alfanuméricos se hace uno a uno, de izquierda a derecha. Si las variables son de diferente longitud pero exactamente iguales hasta el último carácter del más corto, entonces se considera que el más corto es el menor. Solo son iguales dos datos alfanuméricos si son iguales su longitud y sus componentes. Debido al valor numérico en el sistema binario, las letras minúsculas tienen mayor valor que las mayúsculas. Los operadores relacionales son binarios también.

### 2.4.3 Lógicos o booleanos

Combinan sus operandos (*proposiciones simples o compuestas*) de acuerdo con las reglas del álgebra de Boole. El objetivo es producir un nuevo valor (FALSO o VERDADERO) que se convierta en el valor de la expresión. Las condiciones para control de flujo son expresiones de este tipo. Los operadores booleanos que utilizaremos básicamente son:

- **OR** ($\vee$) Suma lógica
- **AND** ($\wedge$) Producto lógico
- **NOT** ($\sim$) Negación

**Operador OR u o**

Es un operador binario, son necesarios dos operandos para producir un resultado. La expresión que se forma producirá un valor VERDADERO cuando al menos uno de sus operandos tenga este mismo valor; de otra forma, el valor de la expresión será FALSO. Es el operador lógico de disyunción.

| p | q | *p or q* |
|---|---|---|
| verdadero | verdadero | verdadero |
| verdadero | falso | verdadero |
| falso | verdadero | verdadero |
| falso | falso | falso |

**Operador AND o y**

Es también un operador binario . La expresión formada tendrá valor VERDADERO cuando ambos operandos tengan este mismo valor; en caso contrario, la expresión tendrá valor FALSO. Es el operador lógico de conjunción.

| p | q | *p and q* |
|---|---|---|
| verdadero | verdadero | verdadero |
| verdadero | falso | falso |
| falso | verdadero | falso |
| falso | falso | falso |

**Operador NOT o negación**

Es un operador unario, es decir, solo afecta a un operando. Afecta a una variable o a una expresión cambiando su estado lógico: si es VERDADERO, lo transforma en FALSO, y viceversa.

| p | $\sim p$ |
|---|---|
| verdadero | falso |
| falso | verdadero |

### 2.4.4 Operador de Asignación

Mediante este operador, "←", se almacenan valores en una variable. Estos valores pueden ser constantes o ser resultado de una expresión. La sintaxis de la operación de asignación es

$$Variable \longleftarrow Expresion$$

Recordemos que el tipo de dato de *Variable* debe ser el mismo que el de *Expresión*; en caso contrario se producirá un error de tipo.

De acuerdo con el tipo de dato de *Expresión* hay distintos tipos de asignación.

**Operación de Asignación Aritmética**

Este tipo de asignación corresponde con el esquema

$$Variable \longleftarrow \textit{Expresión Aritmética}$$

El valor de *Expresión Aritmética* puede ser constante, una variable o el resultado de una operación aritmética entre valores constantes o variables. Por ejemplo:

$minuendo \longleftarrow 6$
$sustraendo \longleftarrow (5 \cdot 4) - 7$
$incremento \longleftarrow minuendo$
$incremento \longleftarrow (minuendo + 5)$
$diferencia \longleftarrow (minuendo - sustraendo) + incremento$

A la variable *minuendo* se le asigna un valor constante. La variable *sustraendo* es el resultado de una operación aritmética cuyo resultado es *13*. La variable *incremento* toma el valor de *minuendo* más cinco. El valor de la variable *diferencia* es el resultado de restar *sustraendo* de *minuendo* y sumarle *incremento*.

Las operaciones anteriores se pueden abreviar así:

$diferencia \longleftarrow (6 - ((5 \cdot 4) - 7)) + (6 + 5)$

Una variable puede tomar su valor previo, modificarlo y asignárselo nuevamente:

$diferencia \longleftarrow diferencia + 1$
$diferencia \longleftarrow diferencia * minuendo$

**Operación de Asignación Lógica**

Este tipo de asignación corresponde con el esquema

$$Variable \longleftarrow \textit{Expresión Relacional}$$
$$Variable \longleftarrow \textit{Expresión Lógica}$$

El valor asignado a *Variable* es un valor lógico; puede ser *true*, *false* o el resultado de evaluar una expresión relacional o lógica. Por ejemplo:

$SW \longleftarrow (4 \geq 3)$
$SW2 \longleftarrow 7 \neq 8$
$X \longleftarrow (SW \vee SW2)$
$Y \longleftarrow 9 = 9$

Las variables *SW*, *SW2* y *X* toman valor Verdadero y la variable *Y* toma valor Falso.

## 2.5 Expresiones

En términos generales, una expresión es una relación entre variables y operadores relacionales, aritméticos y/o lógicos.

El valor de una expresión está determinado por el tipo de operadores y operandos que combina. Además, este valor se verá modificado por el orden en el que se realicen la(s) operacion(es) implicadas en la expresión. Del mismo modo que los operadores, las expresiones se clasifican también en aritméticas, relacionales y lógicas.

| Operadores matemáticos | | Operadores lógicos | |
|---|---|---|---|
| $+$ | Suma | *or* | o |
| $-$ | Resta | *and* | y |
| $*$ | Producto | $\sim$ | negación |
| $/$ | División | $>$ | mayor que |
| $\%$ | Residuo | $<$ | menor que |
| $\subset$ | Subconjunto | $=$ | igual |
| $\subseteq$ | Subconjunto o igual | $\leq$ | menor o igual |
| $\wedge$ | Potenciación | $\geq$ | mayor o igual |
| $(\,)$ | Paréntesis | $\neq$ | diferente |
| $[\,]$ | Corchetes | $\Longleftrightarrow$ | Si sólo si |
| $\{\,\}$ | Llaves | $\Longrightarrow$ | Si entonces |

## 2.5.1 Expresiones aritméticas

Se componen de operadores y funciones aritméticas y sus operandos tienen valor de tipo numérico (*variables* y *constantes*); su resultado también es numérico. En algoritmos son utilizadas básicamente para la asignación de valores a las variables.

Por ejemplo:

| Operando 1 | Operando 2 | Expresión aritmética | Resultado |
|---|---|---|---|
| 3 | 4 | $3 + 4$ | 7 |
| 4,78 | 3,91 | $4,78 * 3,91$ | 18,6898 |
| 10 | 6 | 10 MOD 6 | 4 |
| 81 | - | $\sqrt{81}$ | 9 |
| 45 | 8 | 45 DIV 8 | 5 |
| 6 | -8,4 | $6 - 8,4$ | $-2,4$ |

## 2.5.2 Expresiones relacionales o condicionales

Una expresión relacional es utilizada para establecer comparaciones entre operandos lógicos (*variables* y *constantes lógicas*) mediante los operadores relacionales o condicionales. Su resultado es un valor lógico. Por ejemplo:

| Operando 1 | Operando 2 | Expresión relacional | Resultado |
|---|---|---|---|
| 12,01 | 10,53 | $12,01 > 10,531$ | verdadero |
| 45 | 8 | $45 < 8$ | falso |
| 7 | 8 | $7 = 8$ | falso |
| 3,6 | 4 | $3 \geq 4$ | falso |
| 6 | 6 | $6 \leq 6$ | verdadero |
| 5 | 8,9 | $5 \neq 8,9$ | verdadero |

En la comparación de valores reales por medio del operador =, ya sean *float* o *double*, debido a las limitaciones de la computadora con respecto a la precisión, es posible obtener un valor falso incluso si los dos valores reales son exactamente iguales. En algunos lenguajes de programación, aun si los valores almacenados son los mismos, la comparación de igualdad entre ellos dará como resultado un valor falso.

### 2.5.3 Expresiones lógicas o booleanas

Las expresiones lógicas, al igual que las expresiones relacionales, dan como resultado un valor lógico. Son utilizadas en la especificación de condiciones para control de flujo en los algoritmos. Estas expresiones generalmente están compuestas de otras expresiones más simples, cuya evaluación arroja valores lógicos también.

Por ejemplo:

| Operando 1 | Operando 2 | Expresión lógica | R1 | R2 | Resultado |
|---|---|---|---|---|---|
| $(4 > 3)$ | $(4 \neq 2)$ | $((4 > 3) \wedge (4 \neq 2))$ | V | V | V |
| $(7 \leq 8)$ | $(2 > 3)$ | $((7 \leq 8) \wedge (2 > 3))$ | V | F | F |
| $\sim (11 \geq 9)$ | - | $\sim (11 \geq 9)$ | F | - | F |
| $\sim (61 \leq 43)$ | $(76 = 3)$ | $(\sim (61 \leq 43) \vee (76 = 3))$ | V | F | V |
| $(9 = 1)$ | $(9 > 9)$ | $((9 = 1) \vee (9 > 9))$ | F | F | F |

### 2.5.4 Evaluación de expresiones

Para la evaluación de expresiones es necesario definir primero la precedencia de operadores aritméticos y lógicos, la cual consiste en la prioridad y orden en que se realizarán las operaciones contenidas en la expresión. A continuación se especifican:

| Operadores matemáticos | Operadores lógicos |
|---|---|
| $( ), [ ], \{ \}$ | $\sim$ |
| $\wedge$ | $> < \geq \leq$ |
| $\cdot, /, \%$ | $= \neq$ |
| $+, -$ | *and*, *or* |

La tabla anterior indica que las operaciones delimitadas por corchetes, paréntesis o llaves se realizarán primero que las que no están contenidas en estos operadores. El circunflejo (*operador de la potencia*) junto con las expresiones que tengan operadores relacionales tienen la siguiente prioridad en la tabla. Luego las operaciones de multiplicación y división, seguidas de los operadores de comparación de igualdad o desigualdad. Los operadores suma y resta tienen la prioridad más baja en los operadores matemáticos. Finalmente, las expresiones con los operadores lógicos *and* y *or* son evaluadas en último lugar.

**Ejemplos**

Si los valores de las variables $a$, $b$ y $c$ son 5, 3 y 1, respectivamente, y la variable *sw* tiene valor inicial falso, entonces

- $(a \leq b) \wedge (b \geq c)$
  $(5 \leq 3) \wedge (3 \geq 1)$
  $(falso) \wedge (verdadero)$
  $falso$

  **Comentarios:** De acuerdo con el valor de las variables, en la evaluación de condiciones se obtiene *falso* para la primera condición y *verdadero* para la segunda condición. Teniendo en cuenta los valores de la tabla de verdad del operador de conjunción, al evaluar el valor resultante es falso.

- $sw2 \longleftarrow (c \leq b)$
  $sw2 \longleftarrow (1 \leq 3)$
  $sw2 \longleftarrow verdadero$

  **Comentarios:** La variable $sw2$ ha sido declarada de tipo *booleano* y se le asigna el valor resultante de la operación lógica entre $c$ y $b$ con el operador menor o igual. De acuerdo con el valor de las variables, en la evaluación de condiciones se obtiene *verdadero* para esta condición.

- $(2b = a + c)$
  $(2 \cdot 3 = 5 + 1)$
  $(6 = 6)$
  $verdadero$

  **Comentarios:** De acuerdo con el valor de las variables, en la evaluación de condiciones, en esta operación lógica con operador de igualdad se obtiene *verdadero* luego de la evaluación de la condición.

- $\sim sw$
  $\sim falso$
  $verdadero$

**Comentarios:** En este ejemplo se niega el valor de la variable $sw$. De acuerdo con el valor ya definido de la variable, luego de la evaluación de la condición, se obtiene *verdadero*.

- $sw2 \longleftarrow (\sim sw) \wedge (5 > 3)$
  $sw2 \longleftarrow (\sim falso) \wedge (verdadero)$
  $sw2 \longleftarrow verdadero \wedge verdadero$
  $sw2 \longleftarrow verdadero$

  **Comentarios:** A la variable $sw2$ se le está asignando el resultado de la operación lógica de la conjunción que involucra a la negación de la variable $sw$ y a la comparación con el operador mayor. De acuerdo con el valor de las variables, en la evaluación de condiciones se obtiene *verdadero* para la ambas condiciones, por lo tanto, el valor resultante es verdadero.

- $sw2 \longleftarrow (a > b)$
  $sw2 \longleftarrow (5 > 3)$
  $sw2 \longleftarrow verdadero$

  **Comentarios:** A la variable $sw2$ se le asigna el resultado de la operación lógica de la comparación con el operador mayor de $a$ y $b$. De acuerdo con el valor de las variables, en la evaluación de condiciones se obtiene *verdadero* para esta condición:

- $sw2 \longleftarrow \sim ((a > b) \vee (b > c))$
  $sw2 \longleftarrow \sim ((5 > 3) \vee (3 > 1))$
  $sw2 \longleftarrow \sim ((verdadero) \vee (verdadero))$
  $sw2 \longleftarrow \sim (verdadero)$
  $sw2 \longleftarrow falso$

  **Comentarios:** A la variable $sw2$ se le asigna el resultado de la operación lógica de la negación de la disyunción de dos comparaciones: la primera entre $a$ y $b$ con el operador mayor, y la segunda entre $b$ y $c$ con el operador mayor. De acuerdo con el valor de las variables, en la evaluación de condiciones se obtiene *verdadero*; luego de la negación el valor asignado a la variable $sw2$ es *falso*.

- $x_1 \longleftarrow \dfrac{-b + \sqrt{b^2 - 4ac}}{2a}$
  $x_1 \longleftarrow \dfrac{-5 + \sqrt{5^2 - 4 \cdot 5 \cdot 1}}{2 \cdot 5}$
  $x_1 \longleftarrow \dfrac{-5 + \sqrt{25 - 20}}{10}$
  $x_1 \longleftarrow \dfrac{-5 + \sqrt{5}}{10}$

  **Comentarios:** A la variable $x_1$ se le asigna el resultado de la operación matemática correspondiente a la fórmula general para hallar las raíces soluciones

de una ecuación de segundo grado. De acuerdo con la precedencia de operadores, se efectúan las operaciones y se verifica que el discriminante es positivo y, por lo tanto, la raíz que se calculó es real.

- $x_1 \longleftarrow \dfrac{-b - \sqrt{b^2 - 4ac}}{2a}$
  $x_1 \longleftarrow \dfrac{-5 - \sqrt{5^2 - 4 \cdot 5 \cdot 1}}{2 \cdot 5}$
  $x_1 \longleftarrow \dfrac{-5 - \sqrt{25 - 20}}{10}$
  $x_1 \longleftarrow \dfrac{-5 - \sqrt{5}}{10}$

  **Comentarios:** A la variable $x_1$ se le asigna el resultado de una operación matemática en la que se incluye una raíz cuadrada. Se evalúa el contenido de la raíz, y su valor es positivo, por lo tanto queda una división de $b$ entre un número real.

Evalúe las dos últimas expresiones aritméticas con los siguientes conjuntos de valores:

- $(a = 2, b = 4, c = 9)$
- $(a = 5, b = 4, c = 10)$

Establezca las diferencias de las siguientes expresiones con la expresión aritmética usada en el punto anterior:

- $x_1 \longleftarrow \dfrac{-b + \sqrt{(b^2 - 4) \cdot ac}}{2a}$
- $x_1 \longleftarrow \dfrac{-b}{2a} + \frac{\sqrt{b^2-4ac}}{2a}$

## 2.6 Ejercicios propuestos

1. Realice las conversiones necesarias aplicando las equivalencias de múltiplos y submúltiplos del Byte, según se indique:

   *a*) ¿Cuántos bytes hay en 2378 gigabytes?
   *b*) ¿Cuántas doblepalabras hay en 48 terabytes?
   *c*) ¿Cuántos nibbles hay en 67 kilopalabras?
   *d*) ¿Cuántos discos compactos (CD) son necesarios para guardar 15 gigabytes? ¿Cuánto espacio no se utiliza luego de guardar la información?
   *e*) ¿Cuántos kilobytes hay en 65 900 megapalabras? ¿Cuántos megabytes?

*f*) Se tienen dos discos duros, uno con capacidad de almacenamiento de 80 GB y otro con capacidad de almacenamiento de 0.078125 TB, ¿cuál de los dos almacenará más cantidad de palabras?

*g*) ¿Cuántos bits tiene un conjunto de memorias RAM (memoria principal) si su capacidad conjunta es de 4.5 GB?¿Cuántas megapalabras hay en este conjunto?

*h*) En una compañía se desea almacenar 870 400 MB de información para uso posterior. Se dispone de los siguientes dispositivos de almacenamiento:

- 4 discos duros de 250 GB de capacidad cada uno.
- 50 CD-ROM de 700 MB cada uno.
- 25 DVD-ROM de 4.7 GB cada uno.
- 100 disquetes de 1.44 MB cada uno.

Debe escoger la mejor combinación (*no desaprovechar capacidad de almacenamiento*) de dispositivos para guardar la información. No se tendrá en cuenta la confiabilidad ni el tiempo de vida útil de los dispositivos.

2. Evalúe paso a paso las siguientes expresiones, dando valores aleatorios a cada una de las variables. En caso de expresiones lógicas deberá obtener FALSO o VERDADERO. Tenga en cuenta la precedencia de los operadores y establezca diferencias donde se indique:

*a*) $S^2 = \sqrt{\dfrac{(X-m)^2}{n-1}}$

*b*) $S^2 = \sqrt{\dfrac{X^2-m^2}{n-1}}$ Compare con la expresión anterior.

*c*) $P = \dfrac{-(y^3-1)}{(y+1)-\sqrt{y+1}}$

*d*) $Z = \dfrac{x(x^2+1)^3}{\sqrt{2x}+1}$

*e*) $T = 1 - \dfrac{\sqrt[n]{x-2}}{x^3}$

*f*) $sw \longleftarrow ((x \neq y) \wedge (x \leq y))$

*g*) $sw \longleftarrow ((a \geq b) \vee (b \geq c))$

*h*) Para cada caso indique los errores que existen en la declaración y/o asignación de variables:

1) **Booleano** : $sw \longleftarrow verdadero$
2) **Real** : $valor \longleftarrow verdadero + 267,4$
3) **Real** : $nota \longleftarrow 3,2$
4) **Entero** : $cont2 \longleftarrow 3,2 + 1,5$
5) **Entero** : $6754626$
6) **Entero** : $166numero$
7) **Real** : $sum@3445$
8) **Entero** : $7filcol9$
9) **Entero** : $suma \longleftarrow A$
10) **Real** : $promedio \longleftarrow 98.54331$

# Capítulo 3

# Primitivas algorítmicas

Se utilizan en la solución de un problema en forma de algoritmo para posteriormente ser codificado de acuerdo con la sintaxis de un lenguaje de programación. En los algoritmos se usan palabras y frases del lenguaje natural sujetas a unas determinadas reglas. Todo algoritmo consta básicamente de un conjunto de primitivas, las cuales se pueden clasificar en la siguiente forma:

- Primitivas de Inicio y Fin
- Primitivas de Asignación
- Primitivas de entrada / salida
- Primitivas condicionales no repetitivas
- Primitivas condicionales repetitivas

De igual forma, debe permitir la declaración de datos, tipos de datos, constantes, variables, expresiones, archivos y cualquier otro objeto que pueda ser utilizado en un programa de computador.

## 3.1 Estructuras de entrada/salida y asignación

Las estructuras básicas para la construcción de la lógica de control algorítmico que desempeñan la función básica de entrada de entrada y salida al computador. Estas se identifican por acciones de lectura (*Lea*) de datos que ingresan al computador sobre una o un conjunto de variables y por la acción de escritura (*Escriba*) de las informaciones derivadas del procesamiento aritmético/lógico del computador, las cuales son almacenadas en un conjunto de variables. Las informaciones derivadas también son acompañadas de literales encerrados entre comillas; por ejemplo: **Escriba**("Resultados de la función $f(x) =$ "), que sirven para identificar los comentarios de los resultados que se le presenta al usuario junto con las variables de salida.

La sintaxis de la estructura de lectura, que sirve para entrar datos al computador mediante un dispositivo de lectura, es:

**Lea** $V_1, V_2, \ldots, V_i, \ldots, V_n$, donde $V_i$ son variables para $1 \leq i \leq n$.

---

Ejemplo 3.1 Lectura de datos

---

Lea las variables de base y altura para calcular el área de un rectángulo.

*Análisis*: El área ($A$) de un rectángulo se define matemáticamente por la expresión matemática $A = b \times h$, donde $b$ es la base y $h$ es la altura de la estructura geométricas el área. La expresión del área ($A$) no se puede calcular sino se tienen la base y la altura; entonces, la notación en algoritmos de la lectura de las variables necesarias para calcular es:

**Lea** $base$
**Lea** $altura$

Lo cual es equivalente a las instrucciones algorítmicas

**Lea** $base, altura$
**Lea** $b, h$ (siendo las variables $b$ igual a la base y $h$ la altura).

---

La estructura básica de escritura, la cual permite presentar resultados de los datos procesados, los cuales al ser calculados por el computador se convierten en información de resultado para el usuario, especificada en su notación es:

**Escribir** $V_1, V_2, \ldots, V_n$

---

Ejemplo 3.2 Escritura de informaciones

---

Escriba la variable $area$, resultado del cálculo del valor del área del rectángulo.

*Análisis*: Recordando que la fórmula $A = h \times b$ sirve para calcular el área del rectángulo, entonces una vez el usuario ha leído los valores de la altura ($h$) y la base ($b$), y calculada la expresión matemática del área en la variable $Area_Resultado \leftarrow b \times h$ (donde $\times$ es el operador de la multiplicación), el valor resultado del área se escribe de las siguientes formas:

**Escribir** $Area_Resultado$ // o en su lugar, mediante la estructura algorítmica:

**Escribir** "El área del rectángulo es = ", $Area_Resultado$

En el último caso, el escriba está acompañado del literal encerrado entre comillas. Se aclara al lector que siempre que se utilice los caracteres seguidos // son comentarios que acompañan a las estructuras algorítmicas.

---

La entrada de datos al computador utilizando la estructura de lectura permite la construcción de resultados de información derivados de los procesos de cálculos que se hagan sobre los datos de entrada, y que deben ser asignados a nuevas variables de procesamiento, en este caso de salida. Generando de esta forma un sistema de Entrada (E), Proceso (P) y Salida (S), representado por E/P/S, de la máquina de procesamiento electrónica de datos (computador); sistema que hace que las salidas (S) sean datos procesados por computador, lo cual gráficamente se representa en la figura 3.1.

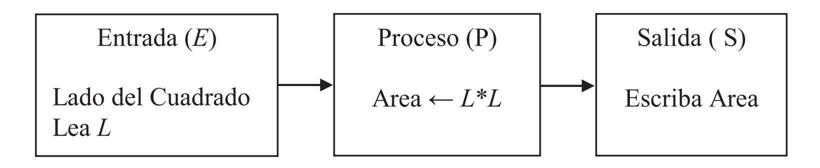



*Figura 3.1* Sistema de proceso E/P/S

---

Ejemplo 3.3 Cálculo del área del cuadrado

---

Diseñe una gráfica representativa de las variables de entrada, el proceso y las salidas que permitan a partir de la lectura de la variable $L$ o lado del cuadrado obtener como resultado el área del cuadrado.

*Análisis*
*Entrada*: Está representada por la variable $L$, dato de entrada o el lado del cuadrado.
*Proceso*: El proceso de cálculo del área es la multiplicación de la variable por sí misma $L \times L$, que da como resultado una nueva variable $Area$, representativa del área del cuadrado $Area = L \times L$ como fórmula matemática; lo cual es representado como estructura algorítmica en la forma $Area \leftarrow L \times L$; ello significa que a la variable $Area$ se le asigna ($\leftarrow$) su lado al cuadrado.
*Salida*: El resultado de la salida es la variable $Area$, la cual contiene la información de cálculo generada por la operación aritmética del producto de de la variable $L$. Gráficamente, el sistema de procesamiento se representa por la figura 3.1.

---

La notación de la estructura básica de asignación, la cual envía el valor calculado de la expresión aritmética o lógica a la variable V, está representada por

$$V \leftarrow \text{expresión}$$

En la cual el valor de la expresión es asignada a la variable $V$, y la expresión está construida con base en operandos y operadores aritméticos o lógicos; en notación algorítmica, dentro del algoritmo se estila escribir únicamente la variable $V$ a la cual

se le asigna ($\leftarrow$) la expresión calculada. Es de la mayor importancia resaltar que la notación algorítmica ($\leftarrow$) implica, con relación a la flecha denotada por Izquierda $\leftarrow$ Derecha, que la parte derecha de la estructura de asignación se calcula y se asigna a la variable de la parte izquierda de la estructura, según se muestra en los ejemplos siguientes:

---

Ejemplo 3.4 Cálculo de expresiones matemáticas

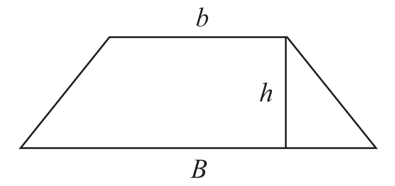
---

Diseñe una expresión algorítmica de asignación para calcular el área de un trapecio.

*Análisis*: Matemáticamente, el área de un trapecio está dada por la base mayor ($B$) más la base menor ($b$) sobre dos, multiplicado por la altura ($h$), lo cual se representa por la fórmula matemática

$$A = \left(\frac{B+b}{2}\right) h$$

Expresión matemática representativa de la siguiente estructura geométrica:




La expresión en notación algorítmica del área del trapecio es $Area_Trapecio \leftarrow ((Base_Mayor + base_menor)/2) \times altura$.

Las variables $Base_Mayor$, $base_menor$ y $altura$ deben haber sido leídas con la estructura algorítmica de entrada Lea.

---

Ejemplo 3.5 Construya una expresión en notación de algoritmos representativa del cálculo de la apotema de un cuadrado

---

*Análisis*: El cálculo aritmético de la apotema de un cuadrado requiere primero la constante numérica del divisor 2 que divide al lado ($L$) del cuadrado. En algoritmo denotada la expresión por

$Apotema_Cuadrado \leftarrow L/2$; // La variable $L$ debe ser leída, y el 2 es una constante.

---

Ejemplo 3.6 Diseñe una expresión algorítmica representativa del volumen de un cono

*Análisis*: El volumen de un cono analíticamente es igual al tercio de su altura multiplicada por el área del círculo de la base; lo que conduce a la expresión matemática

$$A = \frac{1}{3}h\pi r^2$$

Expresión que matemáticamente se representa en notación algorítmica por $Volumen_Cono \leftarrow (3.1416 \times radio \times radio \times altura)/3$.

Se debe tener en cuenta que: *i*) 3.1416 es una constante que representa el valor de $\pi$; y el denominador es también una constante. *ii*) La altura ($h$) y el radio ($r$) son variables que se deben leer. *iii*) Note que la expresión matemática tiene unos paréntesis, que aseguran primero el cálculo del numerador ($3.1416 \times radio \times radio \times altura$), para posteriormente dividir la parte de expresión calculada entre paréntesis por el valor de 3, que representa el denominador.

Las estructuras básicas de control algorítmico de lectura, asignación y escritura permite la construcción de resultados derivados procesos de cálculo con operadores aritméticos o lógicos construidos en la parte de la expresión (parte derecha) para ser asignados a la variable V (parte izquierda del algoritmo).

Las estructuras básicas mencionadas con relación al sistema E/P/S generan un algoritmo cuyo diseño sirve para dar un resultado para el usuario; resultado que es la solución a un problema tratable computacionalmente utilizando la teoría de algoritmos.

La identidad del problema que va a ser solucionado en este libro denota un procedimiento (en este caso un procedimiento principal o **Proc**: (*main*)), el cual tiene un inicio y un final, y como procedimiento tiene las $R_i$ reglas que conforman la lógica del control del algoritmo, lo que equivale al "... paradigma de programación procedimental o estructurado" (García, 2010, p. 5); con la característica de que el procedimiento en su proceso de cálculo debe ser finito en el tiempo; o sea, debe terminar **Fin_proc** para dar una respuesta al usuario.

La notación del procedimiento algorítmico (Proc) en su principio y fin es la siguiente:

**Proc**: Nombre del procedimiento // Equivalente al inicio
| **Lea** $V_1, V_2$ // Variables de entrada
| $V \leftarrow$ Expresión // Expresión lógica o matemática
| **Escriba** $V$ // Variable de salida respuesta del proceso
**Fin_proc** // Equivale al fin del algoritmo

3.1. Estructuras de entrada/salida y asignación

Ejemplo 3.7 Cálculo del área de un triángulo

Diseñar un algoritmo para calcular el área de un triángulo.

*Análisis*: El área de un triangulo está en función de la longitud de su base y su altura y está representada por la expresión aritmética $A = (base \times altura)/2$.

**Proc**: Área del triángulo
**Entero:** *base*, *altura*, *area*
**Lea** *base*, *altura* // Lee los datos de entrada base y altura
$area \leftarrow (base \times altura)/2$ // Calcula el área
**Escriba** "El área del triángulo es ", *area*
// Escribe los resultados del área
**Fin_proc**

Ejemplo 3.8 Cálculo de la longitud de una circunferencia de radio $r$

Diseñe un algoritmo para calcular la longitud de una circunferencia de radio $r$.

*Análisis*: La longitud de una circunferencia de radio $r$ está dada por la ecuación

$$L = 2\pi r$$

El valor de $\pi$ es una constante igual a 3.1416; y la variable del radio ($r$) debe ser leída.

**Proc**: Longitud de circunferencia
**Real:** $r$, $Pi$, $L$
**Lea** $r$ // Lee el radio ($r$) de la circunferencia
$Pi \leftarrow 3.1416$ // Se inicializa la constante $\pi$
$L \leftarrow 2 \times Pi \times r$ // Calcula la longitud de la circunferencia
**Escriba** "La longitud de la circunferencia es igual a", $L$
// Escribe los resultados del cálculo de la longitud $L$ de la circunferencia
**Fin_proc**

Ejemplo 3.9 Cálculo del volumen de un cono

Diseñar un algoritmo para calcular el volumen de un cono teniendo como datos de entrada la generatriz ($g$) del cono y su radio ($r$).

*Análisis*: La generatriz ($g$) del cono en función de la altura del cono ($h$) y de su radio ($r$) es igual a $g^2 = h^2 + r2$; luego $h = \sqrt{g^2 - r^2}$, derivado de lo cual, teniendo la altura, se puede calcular el volumen del cono, el cual es igual a $V = (\pi r^2 h)/3$.

**Proc**: Volumen del cono
**Real:** $g$, $r$, $Pi$, $h$, $V$
**Lea** $g, r$ // Lee los datos de entrada, la generatriz
// y el radio
$Pi \leftarrow 3.1416$ // Se inicializa la constante $\pi$
$h \leftarrow \sqrt{g * g - r * r}$ // Cálculo de la altura ($h$) en función de la
// generatriz y el radio
$V \leftarrow (Pi \times r \times r \times h)/3$ // Calcula del volumen ($V$) del cono
**Escriba** "El volumen del cono es ", $V$
// Escribe el resultado volumen
**Fin_proc**

---

Ejemplo 3.10 Número total de conexiones entre los servidores de una red

Dado un conjunto de $n$ servidores que apoyen una red de computadores, diseñe un algoritmo para calcular el número total de conexiones entre los servidores que soportan la red.

*Análisis*: Un servidor (*server*) es, a su vez, un computador que al estar inserto en una red de computadores suministra servicios de procesamiento de información a otros computadores que reciben el nombre de "clientes". El número de conexiones entre los servidores a través de la red, siendo el número de servidores $n$, está representado por la tabla 3.1.

*Tabla 3.1* Generación del número de conexiones ($C$) derivadas de un conjunto de servidores ($N$)

| Número de servidores ($n$) | Número total de conexiones ($C$) |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 1 |
| 3 | 3 |
| 4 | 6 |
| $\vdots$ | $\vdots$ |
| $N$ | $\binom{N}{2} = \frac{N(N-1)}{2}$ |

**Proc**: Total conexiones entre $n$ servidores
**Entero:** $n$, $c$
**Lea** $n$ // Lee $n$, el número de servidores de la red
$c \leftarrow n \times (n-1)/2$ // Calcula la expresión del número de conexiones
**Escriba** "Para una cantidad de servidores $n =$ ", $n$
// Escribe el número de servidores
**Escriba** "Existe un número total de conexiones = ", $c$
// Número total de conexiones
**Fin_proc**

Ejemplo 3.11 Cálculo del número de metros cúbicos contaminados por una pila

Si es posible que una sola pila contamine 175 000 litros de agua, ¿cuántos metros cúbicos de agua serán contaminados por la población de una ciudad de $n$ millones de habitantes en un año si cada persona utiliza 2 pilas al semestre?

*Análisis*: En los dos semestres del año cada persona utilizará 4 pilas. Luego, el número de litros que contaminan los $n$ millones de personas es de $4 \times n \times 175000$ ($n$ en millones). Por lo tanto, el número de metros cúbicos contaminados ($Mcc$) es igual a $4 \times 175 \times n$, porque un metro cúbico tiene 1000 litros.

**Proc**: Metros cúbicos contaminados
  **Entero:** $n$, $Mcc$
  **Lea** $n$ // Lee en millones $n$, el número de
  // habitantes de la ciudad
  $Mcc \leftarrow 4 \times 175 \times n$ // Calcula la expresión del número de $m^3$
  // contaminados
  **Escriba** "Una ciudad con un número de = ", $n$, " millones de habitantes"
  // Escribe el resultado de la contaminación
  **Escriba** "Contamina = ", $Mcc$, " millones de metros cúbicos de $H_2O$"
  // Si $n = 1$ millón, 700 millones de metros cúbicos de agua
  // son contaminados
**Fin_proc**

Ejemplo 3.12 Cálculo de la resistencia total de un circuito en paralelo

Suponga que un circuito eléctrico tiene dos resistencias en paralelo, $R_1$ y $R_2$, diseñe un algoritmo para calcular la resistencia total del circuito.

*Análisis*: La resistencia total de un circuito en paralelo de $n$ resistencias está representada por la ecuación

$$\frac{1}{R_t} = \frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_n}.$$

Luego, la resistencia total del circuito está dada por la fórmula

$$R_t = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_n}}.$$

Por lo tanto, cuando se tienen dos resistores conectados en paralelo, la resistencia total es

$$R_t = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$$

**Proc**: Cálculo de la resistencia total de dos resistores conectados en paralelo
**Real:** $R1$, $R2$, $Rt$
**Lea** $R1, R2$ // Lee los resistores R1 y R2 del circuito
// Calcula la expresión del número de conexiones
$Rt \leftarrow 1/((1/R1) + (1/R2))$
**Escriba** "Para un circuito con dos resistores = ", $R1$, ", ", $R2$
// Escribe el valor de las resistencia conectadas en paralelo R1 y R2
**Escriba** "conectado en paralelo, la resistencia total Rt es igual a = ", $Rt$
// Escribe el cálculo de la resistencia total
**Fin_proc**

---

Ejemplo 3.13 Área del trapecio circular

---

Dado el radio mayor ($R$) y el radio menor ($r$) de un trapecio circular de amplitud en grados igual a g grados, diseñe un algoritmo que calcule el área del trapecio circular.

*Análisis*: La ecuación representativa del área ($A$) del trapecio circular es

$$A = \frac{\pi(R^2 - r^2)g}{360^\circ}$$

**Proc**: Cálculo del área del trapecio circular
// Lee primero el radio mayor ($Rma$), el radio menor ($Rme$) y
// la amplitud en grado ($g$)
// Se debe tener en cuenta diferenciar las variables
// de lecturas de los dos radios
**Entero:** $Rma$, $Rme$, $g$, $Pi$, $A$
**Lea** $Rma, Rme, g$
$Pi \leftarrow 3.1416$ // Se inicializa la constante Pi
$A \leftarrow (Pi \times (Rma \times Rma - Rme \times Rme) \times g)/360$
// Calcula el área del trapecio circular
**Escriba** "Para un trapecio de radio mayor y de radio menor
iguales a = ", $Rma$, ", ", $Rme$
**Escriba** "Si el trapecio tiene una amplitud en grados de = ", $g$
**Escriba** "El área del trapecio circular es igual a = ", $A$
// Escribe el área del trapecio circular
**Fin_proc**

---

Ejemplo 3.14 Cálculo del valor presente

---

Suponga que un inversionista invierte un capital $C$ a una tasa de interés $t$ durante $n$ años. Si el interés de inversión es del 7 %, diseñe un algoritmo que calcule el valor presente del capital invertido al final del año $K$ de la inversión.

*Análisis*: Un capital $C$ que se invierte a una tasa de interés del $t$ a $n$ años al término de los años mencionados recibirá un capital que traído a valor presente ($VP$) es igual a

$$VP = \frac{C}{(1+t)^n}$$

**Proc**: Valor presente de una inversión
**Entero:** $C$, $K$
**Real:** $t$, $VP$
**Lea** $C, K$ // Se lee el capital $C$ y el número de años $K$
$t \leftarrow 7/100$ // Se inicializa la constante del interés
// de inversión
$VP \leftarrow C/(1+t) **K$ // Se calcula el valor presente
// de la inversión en los K años
// La notación ** es el símbolo del operador
// exponencial
**Escriba** "El valor presente de la inversión es = ", $VP$
**Fin_proc**

---

Ejemplo 3.15 Método de ciframiento del Caesar

---

Suponga el sistema de números digitales [0 al 9]; en un sistema de seguridad utilizan el método de cifrado llamado *Caesar Method* con un desplazamiento de $n$ ($1 \leq n \leq 9$); si se captura un dígito fuente igual a $d$, diseñe una algoritmo para cifrar el dígito.

*Análisis*: Los número digitales son 0 1 2 3 4 5 6 7 8 9; si cada uno de ellos se cifra con el método del César con desplazamiento de 1 unidad, los resultados son 1 2 3 4 5 6 7 8 9 10; si a estos números se le aplica el módulo 10, el resultado de la función da un texto cifrado de números igual a 1 2 3 4 5 6 7 8 9 0; luego, si el código es 72815234 si se cifran sus dígitos, con la función de cifrado anteriormente expuesta, da como resultado 83926345; por lo tanto, la función de Encriptación (E) que recibe un dígito $d$ y se desplaza $n$ posiciones de cifrado está representada por la ecuación

$$E_n(d) = (d+n) \,\mathbf{mod}\, 10$$

**Proc**: *Caesar Method* con desplazamiento $n$
**Entero:** $d$, $n$, $c$
**Lea** $d, n$ // Se lee el dígito que se va a cifrar $d$ y el
// desplazamiento aplicado $n$
$c \leftarrow (d+n) \,\mathbf{mod}\, 10$ // cifrado del dígito con desplazamiento
// $n$ sobre la variable $c$
// **mod** es la función módulo
1

1
**Escriba** "El dígito de entrada = ", $d$
**Escriba** "Cifrado con el método del César con un desplazamiento = ", $d$
**Escriba** "El dígito cifrado es igual a = ", $c$
**Fin_proc**

---

## 3.2 Concepto de primitivas básicas para la construcción de algoritmos

La lógica de control de un algoritmo se basa en un conjunto de estructuras lógicas que se ejecutan secuencialmente desde la primera estructura hasta la última con una cantidad de esfuerzo o trabajo de y en una cantidad finita de tiempo.

Luego las estructuras de entrada/salida y asignación son constructos lógicos primarios, que permiten la elaboración de lógicas de programación más complejas. Los constructos lógicos primarios reciben también el nombre de primitivas lógica ($p_i$), no porque sean antiguas, sino porque son la base para la construcción de estructuras lógicas más complejas.

Definición de primitiva lógica ($p_i$): Una primitiva lógica para la construcción de algoritmos es una instrucción lógica que permite la ejecución de una o varias acciones en el computador; acciones que, a su vez, pueden ser otras primitivas lógicas; primitivas que tienen la característica de ordenar acciones al computador, y cuando dichas acciones son codificadas en un lenguaje de programación reciben el nombre de instrucciones de un lenguaje de programación. Como instrucciones de un lenguaje de programación permiten ser ejecutadas en un computador y consumen tanto un espacio de memoria como una cantidad finita de tiempo para su ejecución.

Con base en lo anterior, las estructuras lógicas de entrada/salida y asignación reciben también el nombre de primitiva de entrada/salida y proceso.

---

Ejemplo 3.16 Residuo y cociente de división entera

---

División entera. Dados dos números enteros numerador y denominador, diseñe un algoritmo que dé como resultado el residuo y el cociente de su división entera.

*Análisis*: Suponga dos números enteros que se leen en las variables numerador = 7 y denominador = 2. El residuo de la división entera 7/2 es igual a 1 (residuo de división entera que en el algoritmo se representa por el operador %); y el cociente de la división entera 7/2 es igual a 3.

---

La tabla 3.2 presenta la relación directa entre las primitivas lógicas de control algorítmico para el caso de la División entera con el lenguaje de programación C.

*Tabla 3.2* Relación entre las primitivas lógicas algorítmicas con el lenguaje de programación C

| Algoritmo | Lenguaje de programación C |
|---|---|
| **Proc**: División entera | `void main() // División entera`<br>`{`<br>`int numerador,denominador,residuo,cociente; // Variables enteras int`<br>`cout<<"Digite el valor del numerador  :   \n";` |
| ($p_1$) Lea numerador → | `cin>>numerador;`<br>`cout<<"Digite el valor del denominador    :   \n";` |
| ($p_2$) Lea denominador → | `cin>>denominador;` |
| ($p_3$) residuo ← numerador**%**denominador → | `residuo=numerador%denominador;` |
| ($p_4$) cociente ← numerador/denominador → | `cociente=numerador/denominador;` |
| ($p_5$) Escriba "Residuo = "residuo → | `cout<<"\t Resultado del residuo   =   \t"<<residuo;`<br>`cout<<"\n";`<br>`cout<<"\n";` |
| ($p_6$) Escriba "Cociente ="cociente → | `cout<<"\t Resultado del cociente   =   \t"<<cociente;`<br>`cout<<"\n";`<br>`cout<<"\n";` |
| **Fin_proc.** | `getch();`<br>`}` |

Las primitivas lógicas descritas en la tabla mencionada son:

- Primitivas lógicas de entrada: Son las primitivas de lectura o entrada de datos representas por Lea numerador ($p_1$) y Lea denominador ($p_2$).
- Primitivas lógicas de asignación: Son las estructuras lógicas primitivas representadas por las asignaciones de cálculo del residuo ($p_3$) y del cociente ($p_4$).
- Primitiva lógica de escritura: Son las estructuras de control representadas por las variables que contienen los resultados de las operaciones de asignación: Escriba "Residuo = " residuo ($p_5$) y Escriba "Cociente = " cociente ($p_6$).

## 3.3 Estructura lógica condicional simple

La lógica de programación se basa en estructuras simples, tales como las estructuras de entrada, asignación y salida (desarrolladas en la sección anterior), a las cuales se les agregan estructuras lógicas más elaboradas; una de las cuales es la estructura condicional.

La "condición" se refiere a una decisión que se debe tomar en la lógica de control del algoritmo. La "decisión" implica la solución a una pregunta que se estructura en la condición. Si la resolución a la pregunta es verdadera, entonces se ejecutan las estructuras lógicas o primitivas lógicas de control que se escriben en la condición verdadera del algoritmo.

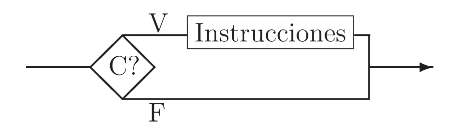



*Figura 3.2* Esquema condicional simple

La estructura condicional simple en función de las primitivas lógicas se define como:

**Condicional Simple**. Sean $p_1, p_2, \ldots, p_n$ un conjunto de primitivas, la condicional simple está representada por

**Si**(condición)**entonces**
$p_1$
$p_2$
$\vdots$
$p_n$
**Fin_si**

Esta condicional permite la ejecución de las primitivas: $p_1, p_2, \ldots, p_n$ si la condición es verdadera o se cumple.

Los ejemplos de estructuras lógicas de programación que se presentan en esta sección corresponden a la primitiva condicional simple; pero se aclara al lector que todos los algoritmos ejemplificados para este caso tienen su contraparte que no es verdadera; contraparte de primitivas por la opción de falso que se desarrollarán con la estructura lógica de control algorítmico identificada como la condicional compuesta.

---

Ejemplo 3.17 Chequeo de un número par

---

Dado un número $n$, el cual es leído, diseñe un algoritmo para chequear si el número es par; y en el caso de que sea par, que imprima un mensaje que diga "número par" e imprima el número que es par.

*Análisis*: Suponga un número dado $n = 8$; entonces, si a 8 se divide entre dos utilizando la división entera $8/2$, la respuesta es 4; ahora, si al resultado de la división entera, o sea, 4, lo multiplicamos por el número dos ($4 \times 2 = 8$), por lo tanto se construye la condición de que si ($\lfloor n/2 \rfloor \times 2 = n$) utilizando la notación de $\lfloor x \rfloor$ para representar la parte entera de $x$, es decir, el mayor entero menor o igual que $x$.

**Proc**: Chequeo de número par
  **Entero:** $n$
  **Lea** $n$ // Leer el número en la variable entera $n$, para decidir si es par
  // Pregunta condicional que chequea si el número dado es par
  **Si**(($\lfloor n/2 \rfloor \times 2 = n$)**Entonces** // Inicio de la estructura condicional Si
    **Escriba** "El número leído es par = ", $n$
    // Primitiva que se ejecuta si la condición es verdadera
  **F.Si** // Cierre del condicional Si
**Fin_proc**

---

Ejemplo 3.18 Conversión de grados centígrados a grados Fahrenheit y a grados Kelvin

---

Diseñe un algoritmo que lea la temperatura en grados centígrados (°C) y la convierta a grados Farenheit (°F) y a grados Kelvin (°K), siempre y cuando la temperatura dada en grados centígrados sea positiva.

*Análisis*: La conversión de los grados centígrados a grados Farenheit y a grados Kelvin se realiza solo cuando el valor de los grados centígrados es mayor que cero; de otro modo no se hace la conversión. Las ecuaciones de conversión son las siguientes:

De grados Farenheit a grados centígrados: $°\text{F} = \frac{9}{5}(°\text{C} + 32)$.

De grados Kelvin a grados centígrados: $°\text{K} = °\text{C} + 273.15$.

**Proc**: Conversor de grados centígrados positivos a Farenheit y Kelvin
**Real:** $C$, $F$, $K$
**Lea** $C$ // Leer la temperatura en grados centígrados (°C)
// Chequea si la temperatura leída es positiva
**Si**($C > 0$)**Entonces**
$F \leftarrow (9/5) \times (C + 32)$
$K \leftarrow C + 273.15$
**Escriba** "La temperatura en grados Centígrados = ", $C$
**Escriba** "Es igual a la temperatura en grados Farenheit = ", $F$
**Escriba** "Es equivalente a la temperatura en grados Kelvin = ", $K$
**F.Si**
**Fin_proc**

---

Ejemplo 3.19 Conversión horaria de 24 horas a 12 horas

---

Dados dos enteros: horas ($h$) y minutos ($m$), representativos del tiempo en horas y minutos dado por un reloj de 24 horas, convierta el tiempo dado por el reloj de 24 horas a un reloj que obtenga el tiempo cada 12 horas.

*Análisis*: En primer lugar se debe verificar si los datos en horas ($h$) y minutos ($m$) dados por el usuario son correctos; luego el rango horario debe estar en el intervalo cerrado [0;24] horas, y para el caso de los minutos, dentro del intervalo [0;60] minutos. En segundo lugar se debe hacer la conversión.

Suponga que los datos de entrada son 17 horas y 57 minutos. Entonces, el equivalente en tiempo horario de 12 horas se calcula de la siguiente forma: se realiza la división entera entre el número de horas dado entre 13; luego $17/13 = 1$ en su cociente; adicionalmente, al cociente calculado se le suma el residuo del módulo 13 aplicado sobre el número de horas recibidas; entonces $17 \,\mathbf{mod}\, 13$ (en lenguaje C $h\,\%13$) es igual 4. Por lo tanto, el tiempo en horas convertido a 12 horas es igual $5 = 17\,\%13 + 17/13$ (siendo esta división entera) $= 4 + 1 = 5$, lo cual es válido en el sentido de que las 17 horas corresponde a las 5 horas.

En el caso de que el tiempo leído en minutos sea igual a 60 minutos, el algoritmo diseñado considera 1 hora, y se la suma al tiempo horario, colocando los minutos a cero.

**Proc**: Conversor horario de 24 horas a 12 horas
**Entero:** $h$, $m$, $Tiempo12$
**Lea** $h, m$ // Leer las horas y los minutos en el reloj de 24 horas
**Si**((($h \geq 0$) y ($h \leq 24$))**Entonces** // Condicional compuesta
// con el operador y
$Tiempo12 \leftarrow (h/13) + (h \,\mathbf{mod}\, 13)$ // División entera, **mod** = %
1 2

1 2

**Si**(($m \geq 0$) y ($m \leq 60$))**Entonces**
  **Si**($m = 60$)**Entonces**
    $Tiempo12 \leftarrow Tiempo12 + 1$
    $m \leftarrow 0$
  **F.Si**
**F.Si**

**Escriba** "El tiempo dado de ", $h$, " horas y ", $m$, " minutos"
**Escriba** "Equivale a = ", $Tiempo12$, " horas y ", $m$, " minutos"
**F.Si**
**Fin_proc**

---

Ejemplo 3.20 Ubicación del semestre de un estudiante en función del número de créditos

---

Un programa académico de pregrado tiene un total de 150 créditos; suponga que por semestre el alumno puede cursar 15 créditos; si $C$ es el número de créditos totales aprobados por el estudiante, diseñe un algoritmo que calcule el semestre de ubicación del estudiante, validando que el número de créditos leídos esté en el intervalo cerrado [1;150] créditos.

*Análisis*: Se debe primeramente verificar si el número de créditos aprobados por el alumno ($C$) está en el rango válido del intervalo cerrado [1;150] créditos. Luego se deben dividir los créditos totales aprobados por el alumno ($C$) entre el número de créditos que puede cursar por semestre, que son 15, y el cociente de la división entera es el semestre en el cual está ubicado el alumno.

**Proc**: Ubicación del semestre de un alumno en función del número de créditos aprobados
  **Entero:** $C$, $Se$
  **Lea** $C$ // Leer el número de créditos aprobados por el alumno.
  **Si**(($C \geq 1$) y ($C \leq 150$))**Entonces** // Validar créditos aprobados
    $Se \leftarrow C/15$ // División entera que calcula el semestre
    **Escriba** "El alumno con ", $C$, " créditos aprobados"
    **Escriba** "Está ubicado en el semestre ", $Se$
  **F.Si**
**Fin_proc**

---

Ejemplo 3.21 Raíces reales de una ecuación cuadrática

---

Diseñe un algoritmo para calcular las raíces reales de una ecuación cuadrática.

*Análisis:* La ecuación cuadrática está representada por $ax^2 + bx + c = 0$, y su solución es la expresión $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. El discriminante de la solución de la ecuación cuadrática es $b^2 - 4ac$; luego, si el discriminante es positivo, las raíces son reales; de otro modo son imaginarias.

**Proc**: Cálculo de las raíces reales de una ecuación cuadrática
  **Real:** $a$, $b$, $c$, $D$, $R1$, $R2$
  **Lea** $a, b, c$ // Leer los coeficientes de la ecuación cuadrática
  $D \leftarrow b * b - 4 * a * c$ // Cálculo del discriminante
  **Si**$(D \geq 0)$**Entonces** // Validar si el discriminante es positivo
    $R1 \leftarrow (-b + \sqrt{D})/(2 * a)$ // Calcular la primera raíz de la ecuación
    $R2 \leftarrow (-b - \sqrt{D})/(2 * a)$ // Calcular la segunda raíz de la ecuación
    **Escriba** "Las raíces de ecuación cuadrática con los coeficientes ",
    **Escriba** $a$, ", ", $b$, " y ", $c$, " son las siguientes: ", $R1$, " y ", $R2$
  **F.Si**
  // Se aclara al lector que hay una contraparte cuando $D < 0$,
  // para la cual se requiere la condicional compuesta, la cual
  // se desarrolla en la siguiente sección
**Fin_proc**

## 3.4 Estructura lógica condicional compuesta

Teniendo como bases iniciales para la construcción de algoritmos las estructuras básicas de control de lectura, asignación, escritura y condicional simple, la estructura condicional compuesta amplía las posibilidades de la lógica de control algorítmica.

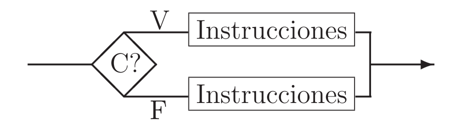



*Figura 3.3* Esquema Condicional Compuesto

La decisión de control en la condicional simple permitió la ejecución de las primitivas cuando se cumple la pregunta en su condición de verdad, y la condicional compuesta permite la ejecución de estructuras de control que se diferencian en el cumplimiento de la condición de la siguiente forma: cuando la condición es verdadera, solo se ejecutan las primitivas de control diseñadas en la parte verdadera de la condición $(p_1, p_2, p_3, \ldots, p_i, \ldots, p_n)$; y cuando la condición es falsa se ejecutan solo las estructuras de control diseñadas en la parte falsa de cumplimiento de la condición $(q_1, q_2, q_3, \ldots, q_i, \ldots, q_n)$, generándose, por lo tanto, dos conjuntos disyuntos de primitivas tales que $(p_1, p_2, \ldots, p_n) \cap (q_1, q_2, \ldots, q_n) = \emptyset$ ($\emptyset$, símbolo de vacío). Lo cual implica que el flujo de control del algoritmo si se ejecuta en el cumplimiento de

verdad de la condición, no se ejecuta en el cumplimiento de la condición falsa; caso que implica que los dos grupos de primitivas $(p, q)$ son disyuntas en la ejecución del algoritmo.

La notación algorítmica de la estructura lógica condicional compuesta es: Dados dos conjuntos disyuntos de primitivas $p_1$, $p_2$, ..., $p_n$ y $q_1$, $q_2$, ..., $q_n$, la condicional compuesta representada por

**Si**(condiciones)**Entonces**
| $p_1, p_2, \ldots, p_n$
**Sino**
| $q_1, q_2, \ldots, q_n$
**F.Si**

permite la ejecución del conjunto de primitivas $p_1$, $p_2$, ..., $p_n$, si la condición es verdadera, o la ejecución del conjunto $q_1$, $q_2$, ..., $q_n$, en el caso de que la condición sea evaluada como falsa.

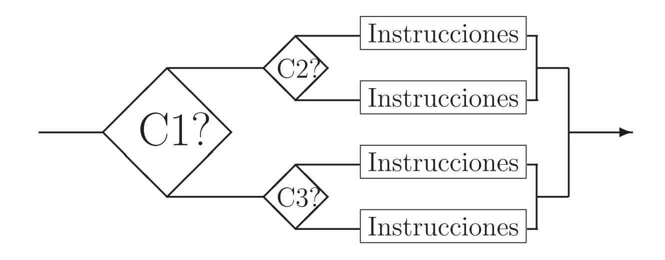



*Figura 3.4* Estructuras condicionales compuestas (anidados)

---

Ejemplo 3.22 Chequeo de un número par o impar

---

Dado un número $n$, diseñe un algoritmo para clasificar el número como par o impar.

*Análisis*: El algoritmo fue analizado y diseñado en el ejemplo 3.17; por lo tanto, se completará correspondiente a la condición falsa del algoritmo cuando el número es impar.

**Proc**: Chequeo de número par o impar
| **Entero:** $n$
| **Lea** $n$ // Leer el número en la variable entera $n$, para decidir si es par
| **Si**($\lfloor n/2 \rfloor * 2 = n$)**Entonces**
| | **Escriba** "El número leído es par"
| | // Primitiva que se ejecuta si la condición es verdadera
1 2

1 2
**Sino**
**Escriba** "El número es impar"
// Primitiva que se ejecuta si la condición es falsa
**F.Si**
**Fin_proc**

---

Ejemplo 3.23 Cálculo del valor absoluto de un número $n$

Diseñe un algoritmo para calcular el valor absoluto de un número leído $n$.

*Análisis*: Si el número es positivo, el valor absoluto del número es el mismo; por otra parte, si el número es negativo, su valor absoluto es el número multiplicado por $-1$; así $|-5| = (-1) * (-5) = 5$.

**Proc**: Cálculo del valor absoluto de $n$
**Entero:** $n$, $a$
**Lea** $n$
// Pregunta condicional que chequea si el número dado es mayor
// o menor que cero
**Si**($n < 0$)**Entonces**
$a \leftarrow -1 * n$
// Primitiva que se ejecutan si la condición es verdadera
**Sino**
$a \leftarrow n$
// Primitiva que se ejecutan si la condición es falsa
**F.Si**
**Escriba** "Para el número leído $n =$ ", $n$
**Escriba** "Su valor absoluto es ", $a$
**Fin_proc**

---

Ejemplo 3.24 Clasificación de un número dado en la recta real

Dado un número leído $X$, el cual está ubicado en la recta real, diseñe un algoritmo para clasificar si el número es positivo, negativo o igual a cero.

*Análisis*: La recta real en el intervalo $(-\infty, +\infty)$ clasifica los números en tres subintervalos: el negativo $(-\infty, 0)$, el positivo $(0, +\infty)$ y el igual a cero $(0)$.

**Proc**: Clasificación del número $X$ en la recta real
**Real:** $X$
**Lea** $X$ // Leer el número que se va a clasificar en la recta real.
// Pregunta condicionales de clasificación del número en la recta real
1

1

**Si**$(X < 0)$**Entonces** // Clasifica el número como negativo
**Escriba** “El número es negativo ”, $X$
**Sino Si**$(X = 0)$
// Clasifica el número como igual a cero
**Escriba** “El número es igual a cero ”, $X$
**Sino**
// Clasifica el número como positivo
**Escriba** “El número es positivo”, $X$
**F.Si**
**Fin_proc**

---

Ejemplo 3.25 Cálculo de una función $F(x)$ por intervalos

---

Sea $F : [0, +\infty) \longrightarrow \mathbb{R}$ tal que

$$F(x) = \begin{cases} x^2 & \text{si } 0 \leq x \leq 2 \\ -x^2 + 4 & \text{si } 2 < x \leq 4 \\ x + 4 & \text{si } x > 4 \end{cases}$$

Diseñe un algoritmo para calcular el valor de la función $F(x)$ para un valor de entrada $x$, el cual es leído.

*Análisis*: La función $F(x)$ genera tres estados de cálculo: $i$) Si el valor leído $x$ pertenece al intervalo cerrado $[0, 2]$, entonces el valor de la función es igual a $F(x) = x^2$. $ii$) Si el valor de la función está en el intervalo $(2, 4]$, entonces el valor de la función es $F(x) = -x^2 + 4$ . Finalmente, en el caso de que el valor de entrada esté en un rango de valores mayores que 4, entonces el valor de la función es $F(x) = x + 4$.

**Proc**: Cálculo de la función matemática $F(x)$ por intervalos
**Real:** $x$, $FX$
**Lea** $x$ // Leer el valor de entrada de la función $F(x)$
// Pregunta condicionales que clasifican F(x) en los intervalos definidos
**Si**$((x \geq 0)$ y $(x \leq 2))$**Entonces**
// Calcula F(x) en el intervalo $[0, 2]$
$FX \leftarrow x * x$
**Escriba** “El valor de la función $F(x)$ es igual a = ”, $FX$
**Sino Si**$((x > 2)$ y $(x \leq 4))$
// Calcula $F(x)$ en el intervalo $(2, 4]$
$FX \leftarrow -x * x + 4$
**Escriba** “El valor de la función F(x) es igual a = ”, $FX$
**Sino Si**$((x > 4))$
// Calcula $F(x)$ en el intervalo $(4, +\infty)$
$FX \leftarrow x + 4$

1 2

1 2
**Escriba** "El valor de la función es = ", $FX$
**F.Si**
**Fin_proc**

---

Ejemplo 3.26 Clasificación de variables en tipos binario, intervalo y proporción

Dadas dos variables numéricas $X$ y $Y$, las cuales son leídas, diseñe un algoritmo para clasificar las variables en los tipos binario, intervalo y proporción.

*Análisis*: Las variables nominales o binarias son aquellas que establecen categorías de clasificación en la escala de medición binaria $(X = 1)$ es diferente de $(Y = 0)$, implican las variables binarias el estado de switch de control, en el caso de su estado activo en uno dejando pasar la corriente, o no activo en cero impidiendo el paso de la corriente. Las variables intervalo, además de contener relaciones de orden $X > Y$ o $X < Y$, permiten establecer la distancia entre los puntos $X$ y $Y$. Es el caso en el cual la edad de María es de 8 años y la de Pedro es de 19; luego, se establece una relación ordinal en la cual se puede concluir que María es menor que Pedro, o lo que es equivalente, $8 < 19$; en el caso de un orden ascendente de orden de edades, o en un sentido equivalente, Pedro es mayor en edad que María, justificado por el hecho de que $19 > 8$ en una relación de orden de edades descendente.

Finalmente, las variables de tipo proporción permiten establecer la razón entre las variables $X$ y $Y$; en tal sentido, $X = k*Y$ o $Y = k*X$, donde $k$ es la constante de proporcionalidad entre las dos variables. El caso se presenta en el sistema solar, si se tiene en cuenta que el número de lunas del Marte $(M)$ son 2 y el número de lunas del planeta Neptuno $(N)$ son 8; entonces se puede concluir que $N/M = 8/2 = 4$, o lo que es equivalente, $N = 4 * M$, o lo que es equivalente, $8 = 4 * 2$.

El algoritmo que se va a diseñar se construye en función de la estructura lógica de control condicional simple, y considera que variables del tipo intervalo también pueden ser clasificadas como variables de tipo proporción. Luego, si $X = 8$ y $Y = 2$, en la clasificación de por tipo intervalo la distancia de $Y$ a $X$ es $(X - Y = 8 - 2 = 6)$; pero en las clasificación tipo razón o proporción, si se calcula la constante de proporcionalidad $k = X/Y = 8/2 = 4$, se puede conceptuar que las variable $X$ es $k$ veces la variable $Y$; o lo que es equivalente, $X = 8 = 4 * 2$, con $Y = 2$ y constante de proporcionalidad $k = 4$.

**Proc**: Clasificador de variables binarias, intervalo y proporción
**Entero:** $X$, $Y$, $D$, $K$
**Lea** $X, Y$ // Leer el valor de las variables que van a ser clasificadas
// Estructura de primitivas condicionales que clasifican las variables
1

1
**Si**$((X = 0 \text{ o } X = 1) \text{ y } (Y = 1 \text{ o } Y = 0))$**Entonces**
  **Escriba** "Las variables pertenecen a la escala binaria"
**Sino Si**$((X \neq 0 \text{ o } X \neq 1) \text{ y } (Y \neq 1 \text{ o } Y \neq 0))$
  // Clasificador intervalo
  **Si**$(X > Y)$**Entonces**
    $D \leftarrow X - Y$
    **Escriba** "Las variables son del tipo intervalo"
    **Escriba** "La distancia de $Y$ a $X$ es = ", $D$
  **Sino Si**$(X < Y)$
    $D \leftarrow Y - X$
    **Escriba** "Las variables son del tipo intervalo"
    **Escriba** "La distancia entre de $X$ a $Y$ es = ", $D$
  **F.Si**
**Sino**
  // Clasificador proporción
  **Si**$(X > Y)$**Entonces**
    $K \leftarrow X/Y$
    **Escriba** "Las variables son del tipo proporción"
    **Escriba** "La variable $X$ es ", $K$, " veces mayor que la variable $Y$"
  **Sino Si**$(Y > X)$
    $K \leftarrow Y/X$
    **Escriba** "Las variables son del tipo proporción"
    **Escriba** "La variable $Y$ es ", $K$, " veces mayor que la variable $X$"
  **F.Si**
**F.Si**
**Fin_proc**

---

Ejemplo 3.27 Intersecto de dos funciones

Dadas dos funciones $F1(x) = x$ y $F2(x) = x^2$. Suponga que se lee un valor $X$, diseñe un algoritmo para identificar si las dos funciones se intersectan en el valor leído.

*Análisis*: Si las dos funciones se intersectan en el argumento de entrada $X$, entonces significa que $F1(x) = F2(x)$. Suponga que $X = 1$, entonces $F1(x) = 1$ y $F2(x) = 1 * 1$; por lo tanto, en el punto 1 las dos funciones son iguales.

**Proc**: Intersecto de funciones en el punto $X$
  **Real:** $x$, $F1$, $F2$
  **Lea** $x$ // Leer el valor para el cual se va a calcular el intersecto
  $F1 \leftarrow x$ // Cálculo del valor de la primera función
  $F2 \leftarrow x * x$ // Cálculo del valor de segunda función
1

1
**Si**($F1 = F2$)**Entonces** // Condicional que indica el intersecto
**Escriba** "La función $F1(x) =$ ", $x$
**Escriba** "La función $F2(x) =$ ", $x * x$
**Escriba** "Se intersectan en la ordenada $Y =$ ", $F2$
**Escriba** "Junto con la abscisa $X =$ ", $x$
// Primitivas que se ejecutan si la condición es verdadera
**Sino**
**Escriba** "Las funciones $F1(x) = x$ y $F2(x) = x * x$"
**Escriba** "NO se intersentan en la abscisa $X =$ ", $x$
**Escriba** "Luego, el valor de las dos funciones para el punto $X$"
**Escriba** "es diferente"
// Primitivas que se ejecutan si la condición es falsa
**F.Si**
**Fin_proc**

Ejemplo 3.28 Equivalencia de una escala cuantitativa a cualitativa

Suponga que la escala de calificaciones de una universidad se muestra en la tabla 3.3.

*Tabla 3.3* Escala de calificaciones de una universidad

| **Numérica** | **Cualitativa** |
|---|---|
| $[4.6, 5.0]$ | Excelente |
| $[4.1, 4.5]$ | Muy bueno |
| $[3.6, 4.0]$ | Bueno |
| $[3.3, 3.5]$ | Aceptable |
| $[3.0, 3.2]$ | Aprobado |
| $[2.6, 2.9]$ | Deficiente |
| $[2.1, 2.5]$ | Malo |
| $[0.0, 2.0]$ | Por mejorar y considerablemente |

Diseñe un algoritmo que escriba la equivalencia cualitativa de una nota numérica leída $n$, considerando que el alumno aprobó la calificación.

*Análisis*: Como el estudiante aprobó la calificación, entonces la nota debe ser superior a 3.0. Una vez se verifique que la nota es superior a 3.0, entonces se clasifica en el respectivo intervalo, utilizando estructuras lógicas condicionales compuestas.

**Proc**: Conversor de escalas aprobatorias de calificaciones
**Real:** $n$
**Lea** $n$
// Condicionales que cualifican la nota del alumno
**Si**($n \geq 3.0$)**Entonces** // La nota es válida como aprobada
1 2

```
1 2
    Si((n ≥ 4.6) o (n ≤ 5.0))Entonces
      Escriba "Estudiante Excelente"
    Sino
      Si((n ≥ 4.1) o (n ≤ 4.5))Entonces
        Escriba "El alumno es Muy Bueno"
      Sino
        Si((n ≥ 3.6) o (n ≤ 4.0))Entonces
          Escriba "El estudiante es Bueno"
        Sino
          Si((n ≥ 3.3) o (n ≤ 3.5))Entonces
            Escriba "El alumno registró un desempeño Aceptable"
          Sino
            Si((n ≥ 3.0) o (n ≤ 3.2))Entonces
              Escriba "Alumno Aprobado"
            F.Si
          F.Si
        F.Si
      F.Si
    F.Si
  Sino
    Escriba "El alumno registra una calificación no aprobatoria"
  F.Si
Fin_proc
```

---

Ejemplo 3.29 Ubicación de coordenadas en los cuadrantes del plano cartesiano

Dados dos puntos $X$ y $Y$, que son las coordenadas del plano cartesiano, haga un algoritmo para clasificar el cuadrante del plano donde está ubicada la coordenada $(X, Y)$.

*Análisis*: Los ejes del plano cartesiano clasifican los puntos $X$ y $Y$ en cuatro cuadrantes: 1) Si $X > 0$ y $Y > 0$, entonces la coordenada está en el primer cuadrante; 2) Si $X < 0$ y $Y > 0$, la coordenada pertenece al segundo cuadrante; 3) en el caso de que $X < 0$ y $Y < 0$, la coordenada perteneces al tercer cuadrante; y 4) adicionalmente, si $X > 0$ y $Y < 0$, entonces la coordenada pertenece al cuarto cuadrante. Finalmente, si $X = Y = 0$, la coordenada está ubicada en el origen; de otro modo la coordenada está situada en uno de los ejes cartesianos.

```
Proc: Clasificador de la coordenada (X, Y) en el plano cartesiano
  Real: X, Y
  Lea X, Y  // Leer las coordinadas X y Y a clasificar en el
  // plano en mención
  // Estructura de primitivas condicionales que clasifican la coordenada
1
```

```
1
  Escriba "La coordenada (" X, ",", Y, ")"
  Si(X > 0 y Y > 0)Entonces
    Escriba " está en el primer cuadrante"
  Sino
    Si(X < 0 y Y > 0)Entonces
      Escriba " está en el segundo cuadrante"
    Sino
      Si(X < 0 y Y < 0)Entonces
        Escriba " está en el tercer cuadrante"
      Sino
        Si(X > 0 y Y < 0)Entonces
          Escriba " está en el cuarto cuadrante"
        Sino
          Si(X = 0 y Y = 0)Entonces
            Escriba " está en el origen"
          Sino
            Si(X = 0)Entonces
              Si(Y > 0)Entonces
                Escriba " está en el eje de las Y positivo"
              Sino
                Escriba " está en el eje de las Y negativo"
              F.Si
            Sino
              Si(X < 0)Entonces
                Escriba " está en el eje de las X negativo"
              Sino
                Escriba " está en el eje de las X positivo"
              F.Si
            F.Si
          F.Si
        F.Si
      F.Si
    F.Si
  F.Si
Fin_proc
```

---

Ejemplo 3.30 Orden de crecimiento de funciones para grandes valores de $n$

Dadas dos funciones $F(x) = 2^x$ y $G(x) = x^x$, diseñe un algoritmo que compruebe cuál de las dos funciones es mayor para un valor grande de $x$ tal que $x$ supere a 1 000 000.

*Análisis*: Los cálculos de funciones para grandes valores de $n$ son importantes, pues indican la tendencia de crecimiento de la función, lo cual sirve para realizar el análisis asintótico de las funciones. Luego, para el valor dado de $x$ se calcula el valor de las dos funciones y se comparan para decidir cuál de las dos funciones es mayor.

```
Proc: Ordenador asintótico de funciones F(x) y G(x)
| // Leer el valor de X para el cual se van a calcular F(x) y G(x)
| Real: X, FX, GX
| Lea X
| Si(X ≥ 1000000)Entonces // Chequea para grandes valores de X
| | FX ← 2 ∗ ∗X // Cálculo de la función F(x)
| | GX ← X ∗ ∗X // Cálculo de la función G(x)
| | Si(FX ≥ GX)Entonces
| | | Escriba "La función FX es mayor que GX"
| | Sino
| | | Escriba "La función FX es menor que GX"
| | F.Si
| F.Si
Fin_proc
```

Ejemplo 3.31 Operaciones lógicas booleanas

Dadas dos variables booleanas $a, b$, diseñe un algoritmo que imprima el valor de la tabla de verdad de las operaciones OR, AND y NOT de $a$ y $b$, sujeto a una opción de impresión, la cual es leída en la variable $Op$, y que discrimina la impresión así: Si la opción es 1, se imprime la tabla de verdad OR; si la opción es 2, se imprime la tabla de verdad AND; y en caso contrario se imprime la tabla de verdad del operador NOT.

*Análisis*: Las tablas de verdad de las operaciones mencionadas se presentan en la tabla 3.4. Luego, de acuerdo con la opción leída se distribuye el flujo de control del algoritmo para imprimir la respectiva tabla.

*Tabla 3.4* Tabla de verdad de las operaciones booleanas

| **OR** ( + ) | **AND** ( . ) | **NOT** |
|---|---|---|
| $a + b$ | $a \,.\, b$ | Si $a = 0$, entonces $\overline{a} = 1$ |
| $0 + 0 = 0$ | $0 \,.\, 0 = 0$ | Si $a = 1$, entonces $\overline{a} = 0$ |
| $0 + 1 = 1$ | $0 \,.\, 1 = 0$ | |
| $1 + 0 = 1$ | $1 \,.\, 0 = 0$ | |
| $1 + 1 = 1$ | $1 \,.\, 1 = 1$ | |

**Proc**: Impresor de tablas de verdad OR, AND, NOT
**Entero:** $a$, $b$, $Op$, $B$
**Lea** $a, b$ // Leer las variables booleanas
**Lea** $Op$ // Leer la opción del operador booleano
**Si**($Op = 1$)**Entonces** // Se calcula el valor de $a$ OR $b$
  **Si**($a = 0$ o $b = 0$)**Entonces**
    $B \leftarrow 0$ // Valor booleano ($B$) resultado
  **Sino**
    $B \leftarrow 1$
  **F.Si**
  **Escriba** $a$, " OR ", $b$, " es igual a ", $B$
**Sino**
  **Si**($Op = 2$)**Entonces** // Se calcula el valor de $a$ AND $b$
    **Si**($a = 1$ y $b = 1$)**Entonces**
      $B \leftarrow 1$
    **Sino**
      $B \leftarrow 0$
    **F.Si**
    **Escriba** $a$, " AND ", $b$, " es igual a ", $B$
  **Sino**
    **Si**($Op = 3$)**Entonces** // Se calcula el valor de NOT $A$
      **Si**($a = 1$)**Entonces**
        $B \leftarrow 0$
      **Sino**
        $B \leftarrow 1$
      **F.Si**
      **Escriba** "El valor negado de ", $a$, " es igual a ", $B$
    **F.Si**
  **F.Si**
**F.Si**
**Fin_proc**

---

Ejemplo 3.32 Diseño de la ecuación de la línea recta

Dados dos puntos en el plano cartesiano, representados por las coordenadas $(X1, Y1)$ y $(X2, Y2)$, construya un algoritmo para diseñar la ecuación de la línea de recta y escribir si su pendiente es creciente o decreciente.

*Análisis*: La ecuación de la línea recta es $y = mx + b$. En función de las coordenadas dadas, las ecuaciones son $y_1 = x_1 + b$ y $y_2 = x_2 + b$; resolviendo las ecuaciones de la primera restando la segunda se calcula el valor de la pendiente de la recta, que es igual a $m = \dfrac{y_1 - y_2}{x_1 - x_2}$. Si la pendiente es mayor que cero, es creciente la ecuación de la línea recta; de lo contrario es decreciente. Finalmente, la ecuación de la línea recta en función de las coordenadas dadas es

$$y = \frac{y_1 - y_2}{x_1 - x_2}(x - x_1) + y_1. \quad x_1 \neq x_2$$

**Proc**: Diseño de la ecuación de la línea recta en función de dos coordenadas
  **Real:** $X1$, $Y1$, $X2$, $Y2$, $m$
  **Lea** $X1, Y1, X2, Y2$ // Leer las coordenadas de los puntos
  $m \leftarrow (Y1 - Y2)/(X1 - X2)$
  **Escriba** "La ecuación de la línea recta es $y =$ "
  **Si**$(m \neq 0)$**Entonces**
    **Escriba** $m$, "$(x-$", $X1$, ") + ", $Y1$
    **Si**$(m > 0)$**Entonces**
      **Escriba** "La pendiente de la recta es positiva"
    **Sino**
      **Escriba** "La pendiente de la recta es negativa"
    **F.Si**
  **Sino**
    **Escriba** $Y1$
  **F.Si**
**Fin_proc**

---

Ejemplo 3.33 Escoger el mayor de tres números

Dados tres números distintos, $n$, $m$, $k$, los cuales son leídos, diseñe un algoritmo utilizando primitivas lógicas Si compuestas anidadas para seleccionar el mayor de los tres números.

*Análisis*: Suponga que los números son $n = 9$, $m = 7$ y $k = 3$; entonces el algoritmo debe realizar todas las combinaciones de comparación; sin hacer una enumeración exhaustiva, una de las combinaciones es si $n > m$, lo cual implica que $9 > 7$, luego la respuesta es Si; entonces dentro de ese sí preguntamos que Si $(n > k)$, pregunta que corresponde a $9 > 3$, y como la respuesta es verdadera, entonces se concluye que el mayor de los tres números es 9.

En caso de que $n > m$ sea falso, implicaría que $m > n$; entonces se debe preguntar cómo es $m$ con respecto a $k$. Luego, Si $m > k$, se concluye que el mayor valor es $m$; de lo contrario $k$ es mayor que $m$, y por lo tanto, el número mayor es $k$.

**Proc**: Número mayor entre $n$, $m$ y $k$
  **Entero:** $n$, $m$, $k$
  **Lea** $n, m, k$ // Leer los tres números
  // Estructura de primitivas condicionales seleccionan el número mayor
1

1
**Si**($n > m$)**Entonces**
**Si**($n > k$)**Entonces**
**Escriba** "El número mayor es ", $n$
**Sino**
**Escriba** "El número mayor es ", $k$
**F.Si**
**Sino**
**Si**($m > k$)**Entonces**
**Escriba** "El número mayor es ", $m$
**Sino**
**Escriba** "El número mayor es ", $k$
**F.Si**
**F.Si**
**Fin_proc**

---

Ejemplo 3.34 Operaciones sobre la descomposición de un número $n$ de $d$ dígitos

Dado un número $n$ de tres dígitos, diseñe un algoritmo para descomponer el número en sus cifras de unidades ($u$), decenas ($d$) y centenas ($c$) y realizar las siguientes operaciones: $i$) ¿cuántas cifras del número son iguales a cero?, $ii$) ¿cuántas cifras del número son pares?, $iii$) ¿cuántas cifras del número son impares?, $iv$) la suma de las cifras del número y, $v$) la productoria de las cifras del número leído.

*Análisis*: Un número $n = 704$ en el sistema decimal es igual a 4 unidades, 0 decenas y 7 centenas. $i$) El número de cifras iguales a cero ($cc$) del número es 1. $ii$) El número de cifras pares del número ($cp$) es 1, representada por el número 4. $iii$) La cantidad de cifras impares del número es 1, equivalente al número 7. $iv$) La suma de las cifras del número dado es $7 + 0 + 4 = 11$. $v$) Finalmente, el producto de las cifras del número $7 * 0 * 4 = 0$.

**Proc**: Operaciones sobre la descomposición de un número de tres cifras
**Entero:** $n$, $cc$, $cp$, $ci$, $Suma$, $Producto$, $u$, $d$, $c$
**Lea** $n$ // Leer el número que se va a descomponer
// Se inicializan los contadores
$cc \leftarrow 0$ // Contador de cifras iguales a cero
$cp \leftarrow 0$ // Contador de cifras pares
$ci \leftarrow 0$ // Contador de cifras impares
// Inicializar los contadores de sumatoria y productoria de las cifras
$Suma \leftarrow 0$ // La suma de las cifras se inicializa en cero
$Producto \leftarrow 1$ // El producto de las cifras se inicializa en uno
// Descomposición del número en sus cifras
1

1
$u \leftarrow n \,\mathbf{mod}\, 10$ // Unidades del número $n$
$d \leftarrow n/10 \,\mathbf{mod}\, 10$ // Decenas del número , $n/10$ es división entera
$c \leftarrow n/100 \,\mathbf{mod}\, 10$ // Centenas del número, $n/100$ es división entera
// Parte *i)*
**Si**$(u = 0)$**Entonces**
| $cc \leftarrow cc + 1$
**F.Si**
**Si**$(d = 0)$**Entonces**
| $cc \leftarrow cc + 1$
**F.Si**
**Si**$(c = 0)$**Entonces**
| $cc \leftarrow cc + 1$
**F.Si**
// Partes *ii) y iii)*
**Si**$(u \,\mathbf{mod}\, 2 = 0)$**Entonces**
| $cp \leftarrow cp + 1$
**Sino**
| $ci \leftarrow ci + 1$
**F.Si**
**Si**$(d \,\mathbf{mod}\, 2 = 0)$**Entonces**
| $cp \leftarrow cp + 1$
**Sino**
| $ci \leftarrow ci + 1$
**F.Si**
**Si**$(c \,\mathbf{mod}\, 2 = 0)$**Entonces**
| $cp \leftarrow cp + 1$
**Sino**
| $ci \leftarrow ci + 1$
**F.Si**
$Suma \leftarrow u + d + c$ // Cálculo de la suma de las cifras
$Producto \leftarrow u * d * c$ // Cálculo de la multiplicación de las cifras
// Impresión de resultados
**Escriba** "El número de entrada ", $n$, " está compuesto por: "
**Escriba** "Número de unidades: ", $u$
**Escriba** "Número de decenas: ", $d$
**Escriba** "Número de centenas: ", $c$
**Escriba** "Número de cifras cero del número: ", $cc$
**Escriba** "Número de cifras pares del números: ", $cp$
**Escriba** "Número de cifras impares del número: ", $ci$
**Escriba** "Suma de las cifras: ", $Suma$
**Escriba** "Producto de las cifras: ", $Producto$
**Fin_proc**

# 3.5 Estructura lógica Dependiendo De

La estructura condicional compuesta permite la ejecución de un conjunto de primitivas de control ($p_i$), en la que el cumplimiento de la condición de la estructura en mención hace posible la ejecución de las instrucciones entre muchas alternativas; no solo por el cumplimiento verdadero de la condición, punto en el cual se ejecutan las primitivas $p_1$, $p_2$, $p_3$, ..., $p_n$, sino por el cumplimiento de la condición cuando es falsa, en cuyo lugar se procesan las primitivas $q_1$, $q_2$, $q_3$, ..., $q_n$, siendo el conjunto de primitivas disyuntas. Estas alternativas se amplían considerablemente cuando las estructuras condicionales se encuentran anidadas. La condicional 'Dependiendo De' es una alternativa a la condicional compuesta cuando es necesario en la comparación el cumplimiento de la condición con un valor específico. Luego, si se cumple el valor específico, se ejecutan únicamente las primitivas asociadas con la selección de este valor, y adicionalmente, las primitivas asociadas con un valor son completamente diferentes a las asociadas con el cumplimiento de otro valor.

Formalmente, la estructura lógica *control unitario* se denota por Dependiendo_De(opción) o DD(opción). Sea $V_i$ una variable que contiene el cumplimiento de $V_i$ opciones ($1 \leq i \leq n$); suponga que cada una de las opciones tiene asociadas un conjunto de primitivas disyuntas identificadas como $V_1$: $p_1^1$, $p_2^1$, ..., $p_k^1$ para la opción uno; $V_2$: $p_1^2$, $p_2^2$, ..., $p_k^2$ si la opción toma el valor igual a dos; $V_i$: $p_1^i$, $p_2^i$, ..., $p_k^i$, en el caso de que cumpla la variable $V_i$; y finalmente $V_n$: $p_1^n$, $p_2^n$, ..., $p_k^n$ para el cumplimiento de la variable $n$. Al final de la ejecución de primitivas de la opción seleccionada hay un rompimiento de la ejecución del algoritmo (*break*), en cuyo caso el flujo de control del algoritmo termina.

La estructura lógica compuesta DD es útil en el diseño de algoritmos que requieran el diseño de menús de opciones, o diseños que requieran la modularización parcial de la lógica de control del algoritmo. La lógica de programación modular se desarrolla en el cuarto capítulo.

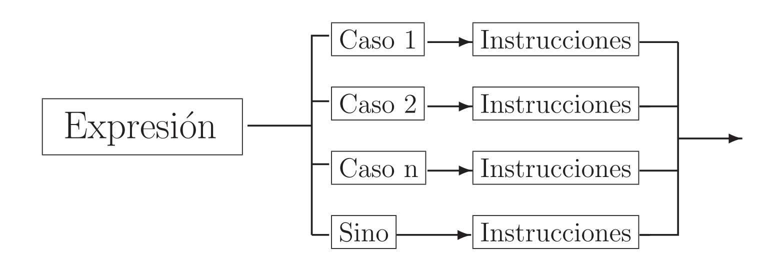



*Figura 3.5* Estructura Dependiendo De

La condicional Dependiendo De en su notación algorítmica se escribe de la siguiente forma:

**Dependiendo_De**(V)**Haga**
  **$V_1$**:
    $p_1^1, p_2^1, \ldots, p_k^1$
  **Fin**
  **$V_2$**:
    $p_1^2, p_2^2, \ldots, p_k^2$
  **Fin**
  ⋮
  **$V_n$**:
    $p_1^n, p_2^n, \ldots, p_k^n$
  **Fin**
**Fin_Dependiendo_De**

La primitiva Dependiendo De (DD) es útil cuando se requiere el diseño de algoritmos con opciones de menús de alternativas, en los que cada alternativa es disyunta y se ejecuta independientemente de las otras.

Ejemplo 3.35 Administrador de funciones trigonométricas

Diseñe un algoritmo que ejecute las funciones trigonométricas $\sin(x)$, $\cos(x)$ y $\tan(x)$, sujetas a la selección de las opciones asociadas así: opción uno para la función Seno, opción dos para la función Coseno y opción tres para el cálculo de la Tangente, siendo el valor de x un argumento de entrada dado en grados.

*Análisis*: Se lee la opción y el valor en grados, se estructura el menú de opciones, y finalmente con el Dependiendo De, sujeto al valor de la opción se ejecuta el cálculo de la función con el argumento ($x$) de entrada.

**Proc**: Administrador de funciones trigonométricas
  **Entero:** $OP$
  **Real:** $x$, $R$
  **Escriba** "Administrador de funciones trigonométricas básicas"
  **Escriba** " 1. Seno"
  **Escriba** " 2. Coseno"
  **Escriba** " 3. Tangente"
  **Escriba** "Por favor digite una de las opciones 1, 2 o 3 únicamente: "
  **Lea** $OP$ // Variable que contiene la opción trigonométrica seleccionada
  **Lea** $x$ // Valor en grado del argumento de entrada de la función
  **DD**($OP$)**Haga**
    **Opción** 1 :
      $R \leftarrow \sin(x)$
      **Escriba** "El valor de la función $\sin(x)$ es igual a ", $R$
    **Fin**
    **Opción** 2 :
      $R \leftarrow \cos(x)$
1 2 3

1 2 3

**Escriba** "El valor de la función cos($x$) es igual a ", $R$

**Fin**

**Opción** 3 :

$R \leftarrow \tan(x)$

**Escriba** "El valor de la función tan($x$) es igual a ", $R$

**Fin**

**Fin_DD**

**Fin_proc**

---

Ejemplo 3.36 Calculadora básica

---

Diseñe un algoritmo que dados dos números, $n$ y $m$, permita el cálculo de la suma, resta, multiplicación, división y exponenciación de los dos números, dependiendo de la selección de opciones en un menú identificadas como: la primera opción para la suma, la segunda para la resta, la tercera para la multiplicación, la cuarta para la división y, finalmente, la quinta para la potenciación $n^m$.

*Análisis*: En primer lugar, la calculadora básica se organiza a través de un menú de acuerdo con las opciones dadas: se lee la opción del menú de la calculadora (supuesta válida en el rango $[1, 5]$); en segundo lugar se leen los números $n$ y $m$; finalmente, de acuerdo con la opción, se ejecuta el cálculo de la operación seleccionada con los argumentos dados.

**Proc**: Administrador de funciones trigonométricas

**Entero:** $OP$

**Real:** $n$, $m$, $R$

**Escriba** "Calculadora Básica"

**Escriba** " 1. Suma"

**Escriba** " 2. Resta"

**Escriba** " 3. Multiplicación"

**Escriba** " 4. División"

**Escriba** " 5. Exponenciación"

**Escriba** "Por favor, digite una de las opciones únicamente: "

**Lea** $OP$ // Contiene la opción que ejecutará la calculadora

**Lea** $n, m$ // Valor de los operandos ($n$ y $m$)

**DD**($OP$)**Haga**

**Opción** 1 :

$R \leftarrow n + m$

**Escriba** "El valor de la suma es igual a ", $R$

**Fin**

**Opción** 2 :

$R \leftarrow n - m$

1 2 3

1 2 3

**Escriba** "El valor de la resta es igual a ", $R$

**Fin**

**Opción** 3 :

$R \leftarrow n * m$

**Escriba** "El valor de la multiplicación es igual a ", $R$

**Fin**

**Opción** 4 :

$R \leftarrow n/m$

**Escriba** "El valor de la división es igual a ", $R$

**Fin**

**Opción** 5 :

$R \leftarrow n * *m$

**Escriba** "El valor de la exponenciación es igual a ", $R$

**Fin**

**Fin_DD**

**Fin_proc**

---

Ejemplo 3.37 Clasificador de alimentos

Suponga que un restaurante ofrece servicios de alimentos a través de Internet. Los tipos de comidas que ofrece son vegetarianas, no vegetarianas y rápidas. Las vegetarianas tienen un menú especificado por sopas de vegetales, ensaladas y jugos. Las no vegetarianas, por bandejas de carne, pollo y cerdo; y las comidas rápidas, de perros calientes y hamburguesas. Diseñe un algoritmo que permita al cliente seleccionar la comida de su preferencia y dar el valor de la compra, suponiendo que el restaurante asigna a través de precios fijos el valor de la compra de cada plato, a lo cual se le debe agregar el IVA de la compra, que es leído. Adicionalmente, si la comida es vegetariana, el restaurante tiene un descuento del 20 %; si es no vegetariana, del 10 %; y si es rápida, del 5 %.

*Análisis*: Se organiza el restaurante en un sistema de menús así: El primer menú son los tipos de comida: 1. Vegetarianas, 2. No vegetarianas, y 3. Rápidas. El segundo menú, dependiendo de la opción seleccionada, se diseña para cada opción un segundo menú dispuesto como: el vegetariano, compuesto por dos opciones: 1. Sopas de vegetales, 2. Ensaladas y jugos. El menú de carnes, compuesto por: 1. Pollo, 2. Res, y 3. Cerdo; y finalmente, el rápido, compuesto por: 1. Perros calientes y 2. Hamburguesas. Posteriormente se solicita al cliente el número de comidas que se va a comprar y se realiza la liquidación de la cuenta.

**Proc**: Sistema de restaurante

**Entero:** *Sopas*, *Ensaladas*, *Pollo*, *Res*, *Cerdo*, *PerrosCalientes*

1

```
1
Entero: Hamburguesas, Iva, OP, OP1, n
Real: V
// Inicialización de precios fijos de menús
Sopas ← 10000
Ensaladas ← 25000
Pollo ← 28000
Res ← 30000
Cerdo ← 33000
PerrosCalientes ← 5000
Hamburguesas ← 7000
Lea Iva
Escriba "Sistema de Restaturante"
Escriba "1. Vegetariano"
Escriba "2. No Vegetariano"
Escriba "3. Comida Rápida"
Escriba "Por favor, digite una de las opciones únicamente: "
Lea OP  // Variable que contiene la opción que ejecutará el restaurante
DD(OP)Haga
  Opción 1 :
    Escriba "1. Sopa Vegetariana"
    Escriba "2. Ensalada y jugo"
    Lea OP1
    Escriba "¿Cuántas comidas quiere comprar?"
    Lea n
    DD(OP1)Haga
      Opción 1 :
        V ← n * Sopas
      Fin
      Opción 2 :
        V ← n * Ensaladas
      Fin
    Fin_DD
    V ← V − 0.2 * V
    V ← V + V * Iva/100
    Escriba V
  Fin
  Opción 2 :
    Escriba "1. Pollo"
    Escriba "2. Carne"
    Escriba "3. Cerdo"
    Lea OP1
    Escriba "¿Cuántas comidas quiere comprar?"
    Lea n
1 2 3
```

```
1 2 3
| | | DD(OP1)Haga
| | | | Opción 1 :
| | | | | V ← n * Pollo
| | | | Fin
| | | | Opción 2 :
| | | | | V ← n * Carne
| | | | Fin
| | | | Opción 3 :
| | | | | V ← n * Cerdo
| | | | Fin
| | | Fin_DD
| | | V ← V − 0.1 * V
| | | V ← V + V * Iva/100
| | | Escriba V
| | Fin
| | Opción 3 :
| | | Escriba "1. Perro Caliente"
| | | Escriba "2. Hamburguesa"
| | | Lea OP1
| | | Escriba "¿Cuántas comidas quiere comprar?"
| | | Lea n
| | | DD(OP1)Haga
| | | | Opción 1 :
| | | | | V ← n * PerroCaliente
| | | | Fin
| | | | Opción 2 :
| | | | | V ← n * Hamburguesa
| | | | Fin
| | | Fin_DD
| | | V ← V − 0.05 * V
| | | V ← V + V * Iva/100
| | | Escriba V
| | Fin
| Fin_DD
Fin_proc
```

Ejemplo 3.38 Simulador de cajero automático

Un sistema de cajero automático tiene cuatro opciones: cambio de clave, consulta de saldos, retiros y transferencias bancarias. Las posibilidades de retiro son: el cliente puede retirar de tres opciones del cajero 500 000, 1 000 000 o cantidad deseada. La opción de transferencia bancaria pide el valor que se va a transferir de la cuenta del cliente a otra cuenta.

Diseñe un algoritmo que simule el sistema de cajero, teniendo en cuenta que para la consulta del saldo y otras transacciones se debe validar la clave del cliente.

*Análisis*: Se organizan dos sistemas de menús: el primero con las cuatro opciones ofrecidas por el banco y el segundo con las opciones de retiro; y en cada una de ella se hace la respectiva acción de la transacción.

```
Proc: Simulador de cajero automático
  Entero: OP, Clave, NuevaClave, Saldo, N, T, STC
  Escriba "Opciones Bancarias"
  Escriba "1. Cambio de Clave"
  Escriba "2. Consulta de Saldo"
  Escriba "3. Retiros"
  Escriba "4. Transferencia de Cuentas"
  Escriba "Por favor, digite una de las opciones únicamente: "

  Lea OP
  DD(OP)Haga
    Opción 1 :
      Escriba "Digite la clave actual"
      Lea Clave
      Si(Clave = Valida)Entonces
        Escriba "Digite Nueva Clave"
        Lea NuevaClave
        Clave ← NuevaClave
      Sino
        Escriba "Clave inválida"
      F.Si
    Fin
    Opción 2 :
      Escriba "Digite la clave actual"
      Lea Clave
      Si(Clave = Valida)Entonces
        Lea Saldo
        // El saldo se encuentra en un dispositivo secundario
        Escriba "El valor del saldo es ", Saldo
      Sino
        Escriba "Clave inválida"
      F.Si
    Fin

    Opción 3 :

1 2 3
```

```
1 2 3
      Escriba "Opciones de Retiro de Efectivo"
      Escriba "1. 500.000"
      Escriba "2. 1'000.000"
      Escriba "3. Cantidad diferente"
      Escriba "Digite la opción de retiro: "
      Lea OP1
      DD(OP1)Haga
        Opción 1 :
          Lea Clave
          Si(Clave = Valida)Entonces
            Lea Saldo
            Saldo ← Saldo − 500000
            Escriba "El valor entregado es 500.000"
            Escriba "El valor del saldo es ", Saldo
          Sino
            Escriba "Clave inválida"
          F.Si
        Fin
        Opción 2 :
          Lea Clave
          Si(Clave = Valida)Entonces
            Lea Saldo
            Saldo ← Saldo − 1000000
            Escriba "El valor entregado es 1'000.000"
            Escriba "El valor del saldo es ", Saldo
          Sino
            Escriba "Clave inválida"
          F.Si
        Fin
        Opción 3 :
          Lea Clave
          Si(Clave = Valida)Entonces
            Lea Cantidad // Cantidad a retirar
            Lea Saldo
            Saldo ← Saldo − Cantidad
            Escriba "El valor entregado es ", Cantidad
            Escriba "El valor del saldo es ", Saldo
          Sino
            Escriba "Clave inválida"
          F.Si
        Fin
      Fin_DD
    Fin
1 2
```

```
1 2
| | Opción 4 :
| | | Lea Clave
| | | Si(Clave = Valida) Entonces
| | | | Lea N  // Número de la cuenta
| | | | Lea T  // Valor que se va a transferir
| | | | Lea Saldo
| | | | Saldo ← Saldo − T
| | | | Lea STC  // Lea saldo de la cuenta
| | | | STC ← STC + T
| | | | Escriba "La transferencia fue realizada correctamente"
| | | Sino
| | | | Escriba "Clave inválida"
| | | F.Si
| | Fin
| Fin_DD
Fin_proc
```

## 3.6 Estructura lógica repetitiva Para

La primitiva de control Para, permite la ejecución de un conjunto de estructuras de control de una forma repetitiva que se ubican dentro del Para, comenzando con un valor inicial, terminando en un valor final, y ejecutando el conjunto de primitivas internas del Para con un incremento de una constante o variable $j$.

La notación algorítmica de la estructura de control en mención es la siguiente:

Dado un conjunto de primitivas $p_1, p_2, p_3, \ldots, p_i, \ldots, p_n$, se escribe como

**Para** $i = 1$ **hasta** $n$ con incrementos de $j$ haga

  $p_1, p_2, \ldots, p_n$

**Fin_Para**

Permite la ejecución repetitiva del conjunto de primitivas $p_1, p_2, p_3, \ldots, p_i, \ldots, p_n$ hasta el valor $n$, o tope del ciclo de control Para, con incrementos de $j$, los cuales afectan la variable índice $i$.

Los ciclos repetitivos en el diseño de algoritmos son útiles en el caso de:

- La generación de secuencias de números.
- Cálculo de sumatorias $\sum$, tales como $\sum_{i=1}^{n} i^2$ o $\sum_{i=1}^{n} f(i)$

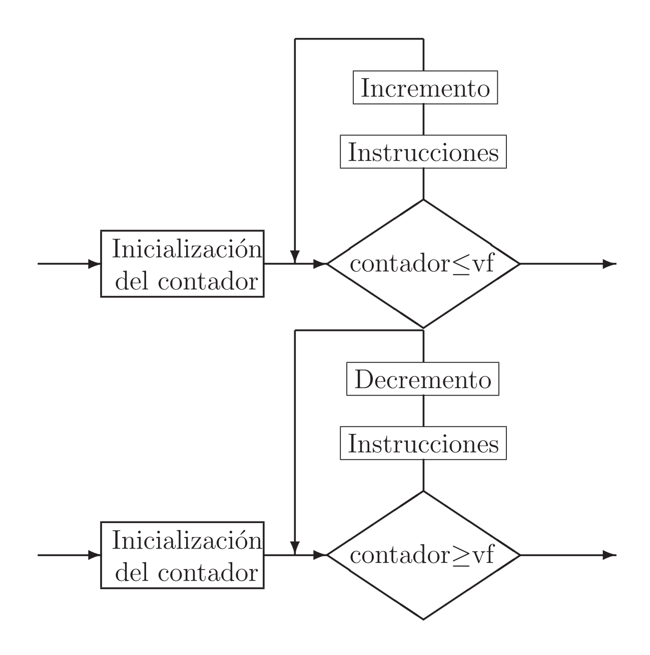



*Figura 3.6* Estructura Repetitiva Para

- Cálculo de series; por ejemplo, la serie de Taylor de la función $f(x) = \sin(x)$,

$$\sin(x) = \frac{x!}{1!} - \frac{x!}{3!} + \frac{x!}{5!} - \cdots = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!}.$$

- Cálculo de productorias $\prod$, como es el caso de $n! = \prod_{i=1}^{n} i = 1 * 2 * 3 * \cdots * n$.
- Generación de series y sucesiones.
- Generación de funciones trigonométricas, entre otros casos, de repetición de un término de una sumatoria, de una productoria, o de una función matemática repetitiva sujeta a la variación de un índice, entre otros casos.
- Acciones de repetición de un conjunto de actividades desde un punto inicial a un punto final, como es el caso de lectura de calificaciones, contadores (luz, agua), censos poblacionales, o encuestas, entre otros; siempre y cundo se tenga el tope o límites del número de actividades que se van a considerar en la repetición (valor de $n$).
- Administración de vectores, y
- Administración de matrices.

Se debe tener en cuenta que el incremento de la estructura Para puede ser positivo o negativo, y se controla con el tope definido dentro de la estructura.

Ejemplo 3.39 Generación de los números naturales

Diseñe un algoritmo para imprimir los $n$ primeros números naturales, validando que el valor de $n$, el cual debe ser leído, es positivo.

*Análisis*: Los números naturales son la generación de los dígitos 1, 2, 3, ..., y sus sucesores hasta el valor de $n$, verificando que el valor de $n$ sea mayor que cero.

**Proc**: Impresor de números naturales hasta el valor de $n$
| **Entero:** $n$, $i$
| **Lea** $n$ // Valor límite o tope de impresión de los números naturales
| **Si**($n \leq 0$)**Entonces**
| | **Escriba** "El número $n$ no es positivo"
| **Sino**
| | **Escriba** "Los números naturales desde el valor 1 hasta el valor"
| | **Escriba** "de $n$ = ", $n$, " son "
| | **Para**($i = 1, n, 1$)**Haga**
| | | **Escriba** "Valor del número natural = ", $i$
| | **Fin_Para**
| **F.Si**
**Fin_proc**

Ejemplo 3.40 Cálculo de la potencia $n$ de un número por multiplicaciones sucesivas

Diseñe un algoritmo para calcular la expresión matemática $r = x^n$ para valores de $x$ y $n$ leídos, suponiendo que $n > 1$, por multiplicaciones sucesivas, validando que el valor de $n$ es positivo.

*Análisis*: La expresión $r = x^n$ por multiplicaciones sucesivas es igual a

$$r = \underbrace{x * x * x * \cdots * x}_{n \text{ veces}}.$$

**Proc**: Potencia del número $n$ por multiplicaciones sucesivas
| **Entero:** $n$, $i$
| **Real:** $x$, $r$
| **Lea** $n$ // Valor límite o tope de cálculo de la máxima potencia de $n$
| **Lea** $x$ // Valor a ser elevado a la potencia $n$
| **Si**($n \leq 0$)**Entonces**
| | **Escriba** "El número $n$ no es positivo"
| **Sino**
| | // Iniciar el resultado de las sucesivas multiplicaciones en la base $x$
| | $r \leftarrow x$
1 2

1 2

**Para**($i = 1, n-1, 1$)**Haga**
$r \leftarrow r * x$
**Fin_Para**
**Escriba** "El valor resultado de ", $x$, " elevado a la potencia ", $n$
**Escriba** "es igual a ", $r$
**F.Si**
**Fin_proc**

---

Ejemplo 3.41 Impresor descendente de números desde el valor de $n$ hasta $k$

Diseñe un algoritmo para imprimir los número desde un valor $n$ hasta un valor $k$ en orden descendente de incrementos igual a $-i$, suponiendo que $n \gg k$.

*Análisis*: Teniendo en cuenta que $n$ es mucho mayor ($\gg$) que $k$, se debe realizar un impresión o escritura de los números desde el valor $n$ hasta el valor $k$ con incrementos de $-i$. Supuesto $n = 15$, $k = 3$ e $i = -3$, entonces los números que se deben escribir son: $15, 12, 9, 6, 3$.

**Proc**: Impresor de números
**Entero:** $n$, $k$, $i$, $j$
**Lea** $n, k, i$
**Si**($n \leq k$)**Entonces**
**Escriba** "Valores de entrada errados porque $n$ debe ser "
**Escriba** "mucho mayor que $k$"
**Sino**
// Ciclo repetitivo que hace la impresión de los números de $n$ hasta $k$
// con incrementos negativos de $i$
**Escriba** "La impresión de los números desde el valor ", $n$
**Escriba** " hasta el valor de ", $k$
**Escriba** " con decrementos de ", $i$, " son "
**Para**($j = n, k, -i$)**Haga**
**Escriba** $j$
**Fin_Para**
**F.Si**
**Fin_proc**

---

Ejemplo 3.42 Cálculo del factorial de número $n$ donde $n \in \mathbb{Z}^+$

Construya un algoritmo para calcular el factorial de un número $n$ que pertenece a los enteros positivos, el cual se debe leer y verificar si pertenece a los enteros positivos.

*Análisis*: La definición matemática de $n!$ es $n = 1 * 2 * 3 * 4 \cdots * (n-1) * n$. Supuesto $n = 7$, entonces el valor del factorial es $1 * 2 * 3 * 4 * 5 * 6 * 7 = 720$.

**Proc**: Cálculo del factorial de un número entero positivo
**Entero:** $n$, $Factorial$, $i$
**Lea** $n$ // Valor límite o tope de cálculo de la máxima potencia de $n$
**Si**($n \leq 0$)**Entonces**
**Escriba** "El número $n$ no es un entero positivo"
**Sino**
// Iniciar el resultado en el módulo de la multiplicación
$Factorial \leftarrow 1$
**Para**($i = 1, n, 1$)**Haga**
$Factorial \leftarrow factorial * i$
**Fin_Para**
**Escriba** "El valor de ", $n$, "! es ", $Factorial$
**F.Si**
**Fin_proc**

---

Ejemplo 3.43 Generación de la serie de los números de Fibonacci y su respectiva suma

---

Diseñe un algoritmo para generar la serie de los números de Fibonacci y su respectiva suma desde el valor 0 hasta el valor $n$, donde $n$, que pertenece a los enteros positivos, es leído.

*Análisis*: La sucesión de Fibonacci ($F_n$) está dada por la fórmula recursiva

$$F_n = F_{n-1} + F_{n-2}. \quad \text{con } F_1 = 1,\ F_2 = 1$$

La serie de Fibonacci corresponde, entonces, a la suma de los $n$ primeros términos de la sucesión, es decir,

$$S_n = \sum_{i=1}^{n} F_i = F_1 + F_2 + \cdots + F_n$$

Por ejemplo, para el valor de $n = 5$ se tiene $F(1) = 1$, $F(2) = 1$, $F(3) = 2$, $F(4) = 3$ y $F(5) = 5$; por lo tanto, la serie de 5 corresponde a la suma $1 + 1 + 2 + 3 + 5 = 12$.

**Proc**: Generación de la serie de Fibonnaci con su respectiva suma
**Entero:** $n$, $Primero$, $Segundo$, $Suma$, $Tercero$
**Lea** $n$ // Valor tope de la serie de Fibonacci
**Si**($n \leq 0$)**Entonces**
**Escriba** "El número no es un entero positivo
**Sino**
$Primero \leftarrow 1$ // Iniciar el primer número de la serie de Fibonacci
$Segundo \leftarrow 2$ // Iniciar el segundo número de la serie de Fibonacci
// Ciclo repetitivo que genera las serie de Fibonacci y su suma
1 2

```
1 2
    Suma ← 1 + 2 // Iniciar la serie de Fibonacci
    Escriba "Sucesión de Fibonacci"
    Escriba "Término 1: ", Primero
    Escriba "Término 2: ", Segundo
    Para(i = 3, n, 1)Haga
        Tercero ← Primero + Segundo
        Suma ← Suma + Tercero
        Escriba "Término ", i, ": ", Tercero
        Primero ← Segundo
        // Ahora el segundo término pasa a ser el primero
        Segundo ← Tercero // y el nuevo pasa a ser el segundo
        Escriba "Suma: ", Suma
    Fin_Para
  F.Si
Fin_proc
```

---

Ejemplo 3.44 Escoger el mayor y el menor número de un conjunto de $L$ números

Sea un conjunto de $L$ números ($L > 0$), los cuales son leídos, diseñe un algoritmo que calcule el número mayor y el número menor del conjunto dado. Asuma que el usuario ingresa un entero positivo $L$.

*Análisis*: Suponga $L = 10$, o conjunto de números cuyos valores son 10, 29, 87, 25, 67, 77, 87, 98, 99, 23. El algoritmo debe sacar el mayor de los números, es decir, 99, y el menor de los números, es decir, el valor de 10.

```
Proc: El mayor y el menor de L números
  Entero: L, i
  Real: n, Mayor, Menor
  Lea L // Leer el valor del número de números a ser leídos
  Lea n // See lee el primer número
  Mayor ← n // Se asigna este número como el mayor
  Menor ← n // Se asigna este número como el menor
  // Al leer más números, se pregunta si es mayor o menor que
  // Mayor y Menor, respectivamente
  L ← L − 1 // Se reduce el valor en 1, debido a que ya se leyó
  // un primer número
  Para(i = 1, L, 1)Haga
    Lea n // See lee el nuevo número
    Si(n > Mayor)Entonces
      // Debido a que se encontró un número mayor que Mayor
      // se actualiza el valor de este último
      Mayor ← n
    F.Si
1 2
```

1 2

**Si**$(n < Menor)$**Entonces**

// Debido a que se encontró un número menor que $Menor$

// se actualiza el valor de este último

$Menor \leftarrow n$

**F.Si**

**Fin_Para**

**Escriba** "El mayor número leído es ", $Mayor$

**Escriba** "El menor número leído es ", $Menor$

**Fin_proc**

---

Ejemplo 3.45 Cálculo del promedio de un conjunto de $n$ números

---

Sea $n$ un conjunto de números, los cuales son leídos, diseñe un algoritmo que calcule el promedio de los $n$ números, verificando que $n \geq 1$ y mostrando el número que va a participar en el promedio.

*Análisis*: Suponga $n = 5$ el conjunto de números a los cuales se les va a calcular el promedio. Si los números dados son 1, 0, 2, 1, y 6, entonces el valor de la suma de los número es igual a 10; luego, el valor del promedio del número de números leídos es $10/5 = 2$.

**Proc**: Cálculo del promedio de un conjunto de números

**Entero:** $n$, $i$

**Real:** $Suma$, $num$, $Promedio$

**Lea** $n$ // Leer el número de números a los cuales se le va

// a calcular el promedio

$Suma \leftarrow 0$ // Iniciar el numerador del promedio, en el módulo

// de la suma

// Ciclo repetitivo que permite leer los $n$ números y acumular su suma

**Para**$(i = 1, n, 1)$**Haga**

**Lea** $num$ // Número que va participar en el promedio

**Escriba** "El número que participa en el promedio es ", $num$

$Suma \leftarrow Suma + num$

**Fin_Para**

// Cálculo del promedio de los $n$ números

$Promedio \leftarrow Suma/n$

**Escriba** "El valor del promedio es igual a ", $Promedio$

**Fin_proc**

---

Ejemplo 3.46 Impresión de las tablas de multiplicación y división del número $n$ hasta $k$

---

Dados dos números $n$ y $k$, pertenecientes a los enteros positivos, diseñe un algoritmo que imprima la tabla de multiplicar del número $n$ hasta el número

$k$, y simultáneamente imprima la tabla de la división de la multiplicación del número $n$ empezando desde $k$ hasta 1, de acuerdo con la siguiente imagen de salida (obviar la verificación de que $n$ y $k$ pertenecen a los enteros positivos):

| | |
|---|---|
| $5 \times 1 = 5$ | $60/12 = 5$ |
| $5 \times 2 = 10$ | $55/11 = 5$ |
| $5 \times 3 = 15$ | $50/10 = 5$ |
| $5 \times 4 = 20$ | $45/9 = 5$ |
| $5 \times 5 = 25$ | $40/8 = 5$ |
| $5 \times 6 = 30$ | $35/7 = 5$ |
| $5 \times 7 = 35$ | $30/6 = 5$ |
| $5 \times 8 = 40$ | $25/5 = 5$ |
| $5 \times 9 = 45$ | $20/4 = 5$ |
| $5 \times 10 = 50$ | $15/3 = 5$ |
| $5 \times 11 = 55$ | $10/2 = 5$ |
| $5 \times 12 = 60$ | $5/1 = 5$ |

*Análisis*: Suponga que va a generar la tabla de la multiplicación del número $n = 5$ hasta el valor de $k = 12$; y simultáneamente se requiere la tabla de la división de la multiplicación de $k * n$, o sea, $12 * 5 = 60$, pero su resultado se divide entre $k$. Se nota que la tabla de la multiplicación va incrementando el valor por el cual se multiplica $n$, y la tabla de la división tiene incrementos negativos desde el valor de $k$ hasta el valor de 1.

```
Proc: Tabla de multiplicación y división
| Entero: n, k, j, i
| Lea n, k
| j ← k  // Valor temporal para controlar la tabla de la división
| Para(i = 1, k, 1)Haga
| | m ← n * i  // Valor de la multiplicación
| | r ← n * j  // Valor resultado, base del numerador de la tabla de división
| | d ← r/j  // Resultado de la división
| | Escriba n, " * ", i, " = ", m
| | Escriba r, " / ", j, " = ", d
| | j ← j − 1  // Se actualiza el valor del contador j
| Fin_Para
Fin_proc
```

Ejemplo 3.47 Simulador de un contador digital

Diseñe un algoritmo para simular un contador digital de unidades (U), decenas (D) y centenas (C) desde el conteo 000 hasta 999, de tal manera que cada vez que se alcance el dígito 9 en las unidades se avance al siguiente dígito en las decenas; y cada vez que se alcance el número 9 en las decenas se avance un dígito más en las centenas hasta completar el tope del contador.

*Análisis*: El simulador del contador parte desde un punto inicial 000 y llega hasta un punto final 999, límite en el cual se han cubierto todos los dígitos de las unidades, de las decenas y de las centenas. Luego inicio implica para el contador que unidades, decenas y centenas inician en ceros 000, luego incrementamos el contador de unidades 001, 002, 003, ..., 009; cuando el contador de unidades llega a 9, es necesario sumar uno a las decenas o sea 010 y seguir contando en las unidades 011, 012, 013, ..., 099; ahora, cuando las decenas lleguen a 9, se necesita aumentar las centenas e iniciar nuevamente las unidades 101, y así sucesivamente.

**Proc**: Simulador de contador digital compuesto por unidades, decenas y centenas

```
    Entero: c, d, u
    Para(c = 0, 9, 1)Haga
        Para(d = 0, 9, 1, )Haga
            Para(u = 0, 9, 1)Haga
                Escriba "Centenas: ", c
                Escriba "Decenas: ", d
                Escriba "Unidades: ", u
            Fin_Para
        Fin_Para
    Fin_Para
Fin_proc
```

Ejemplo 3.48 Cálculo de la varianza muestral de un conjunto de datos

Construya un algoritmo que calcule la varianza muestral de un conjunto de $n$ datos, los cuales son leídos.

*Análisis*: La ecuación de la varianza muestral está representada por

$$S_x^2 = \frac{\sum_{i=1}^{n} x_i^2}{n} - \overline{x}^2$$

con

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}.$$

Luego, $i$) se debe leer el valor del número de datos de entrada ($n$) para el cual se va a calcular la varianza muestral; $ii$) posteriormente se calcula la suma al cuadrado del número de datos leídos y se divide por el número de datos; $iii$) Adicionalmente se tiene que calcular el promedio de los datos ($\overline{x}$, lo cual se lee $x$ barra o $x$ trazo), ello se eleva al cuadrado y se resta del resultado de la operación de $ii$).

**Proc**: Cálculo de la varianza muestral
**Entero:** $n$, $i$
**Real:** $Suma1$, $Suma2$, $x$, $P$, $PSC$, $V$
**Lea** $n$
$Suma1 \leftarrow 0$ // Suma de los datos elevados al cuadrado Xi * Xi
$Suma2 \leftarrow 0$ // Suma de los datos
// Ciclo repetitivo que calcula la varianza muestral
**Para**$(i = 1, n, 1)$**Haga**
**Lea** $x$ // Leer el dato que va a participar en la varianza muestral
$Suma1 \leftarrow Suma1 + x * x$
$Suma2 \leftarrow Suma2 + x$
**Fin_Para**
$P \leftarrow Suma2/n$
$PSC \leftarrow Suma1/n$
// $PSC$ corresponde al promedio de la suma de los cuadrados
$V \leftarrow \sqrt{PSC - P * P}$
// $V$ contiene la varianza
**Escriba** "El valor de la varianza muestral es ", $V$
**Fin_proc**

Ejemplo 3.49 Liquidador de nómina

Diseñe un algoritmo representativo de un sistema liquidador de nómina con las siguientes características: una organización requiere liquidar la nómina de un conjunto de $T$ trabajadores, de los cuales se tiene el NIT del funcionario ($nit$) y su nombre ($nom$). Cada trabajador tiene por mes un básico ($b$) y tiene un número de horas extras diurnas ($hed$), horas extras nocturnas ($hen$) y un valor de recargo nocturno ($vrn$); igualmente, el trabajador tiene horas extras diurnas festivas ($hedf$), horas extras nocturnas festivas ($henf$) y el recargo en días festivos ($vrf$). Los valores de las horas: diurna ($vhd$), nocturna ($vhn$), festiva diurna ($vhfd$) y nocturna ($vhfn$) se deben leer, y son los mismos valores para todos los trabajadores. Adicionalmente, los trabajadores tienen un conjunto de deducciones: deducción por salud ($des$), deducción por pensión ($dep$) y deducción por solidaridad ($del$), siempre y cuando el trabajador gane más de cuatro salarios mínimos legales (SML) sobre el valor básico de ganancia. Teniendo en cuenta que el salario mínimo legal se debe leer ($sml$), el algoritmo liquidador de nómina debe calcular los devengados o ganancias reales ($g$), los deducidos ($d$) y el neto a pagar ($p$), considerando que el recargo nocturno y en días festivos es del 10 % del valor liquidado en horas extras.

*Análisis*: La liquidación de nómina para un NIT ejemplo se muestra a continuación:

- $nit = 10$, $nom = A$, $b = 1000000$

- $hed = 10$, $vhd = 1$, $hen = 0$
- $vrn = 10\,\%$
- $hedf = 0$, $henf = 0$
- $vrf = 10\,\%$
- $des = 20000$, $dep = 10000$, $del = 0$
- $p = 0$

Para cada trabajador el neto a pagar ($p$) es igual a devengados ($g$) menos deducidos ($d$). Luego $p = g - d$. Las ganancias reales o los valores devengados en el caso del trabajador son iguales al básico más los valores devengados en las horas extras ($e$), es decir, $g = b + e$. El valor de las horas extras son iguales a

$$e = (hed * vhd) + (hen * vhn) + (hedf * vhfd) + (hefn * vhfn);$$

y el valor final de las horas extras es $e = e + 0.1 * e$.

Por otra parte, el valor de los deducidos es $d = des + dep + del$. Suponiendo que el valor de la hora diurna es una unidad, entonces el valor neto para el ejemplo es

$$p = 1000000 + 10 * 1 + 10 * 1 * 0.1 - (20000 + 10000) = 970011.$$

**Proc**: Liquidador de nómina
  **Entero:** $t$, $vhd$, $vhn$, $vhfd$, $vhfn$, $i$, $b$, $hed$, $hen$, $hedf$, $henf$
  **Real:** $sml$, $del$, $des$, $dep$, $e$, $g$, $d$, $p$
  **Cadena:** $nit$, $nom$
  **Lea** $t$ // Leer el número de trabajadores para quienes se les
  // liquidará la nómina
  **Lea** $vhd, vhn, vhfd, vhfn$ // Leer los valores de las horas extras
  **Lea** $sml$ // Leer el salario mínimo legal - SML
  **Lea** $del$ // Leer el descuente a aplicar por solidaridad
  // Ciclo repetitivo que calcula la nómina para cada trabajador
  **Para**$(i = 1, t, 1)$**Haga**
    **Lea** $nit, nom$ // Leer el nit y el nombre del trabajador
    **Lea** $b, hed, hen, hedf, henf$ // Leer el básico y número de horas extras
    **Lea** $des, dep$ // Leer deducciones fijas salud y pensión
    // Cálculo extras
    $e \leftarrow (hed * vhd) + (hen * vhn) + (hedf * vhfd) + (henf * vhfn)$
    $e \leftarrow e * 0.1$ // Recargo de horas extras
    $g \leftarrow b + e$ // Ganancias del trabajador
    **Si**$(b > 4 * sml)$**Entonces**
      $d \leftarrow des + dep + del$ // Sumar el descuento de solidaridad

1 2 3

```
1 2 3
│ │ Sino
│ │ │ d ← des + dep // No se suma el descuesto de solidaridad
│ │ F.Si
│ │ p ← g − d // Neto a pagar para el trabajador i
│ │ Escriba "Valor de la nómina para el trabajador ", nit
│ │ Escriba "Nombre: ", nom
│ │ Escriba "Devengados: " g
│ │ Escriba "Deducidos: ", d
│ │ Escriba "Neto (Devengados - Deducidos) a pagar: ", p
│ Fin_Para
Fin_proc
```

Es importante tener en cuenta en el desarrollo del liquidador de nómina que los datos leídos en el ciclo Para son variables temporales que se van actualizando dentro del ciclo; luego, es necesario desarrollar el concepto de estructuras de datos, con el fin de que siendo almacenadas las variables temporales en dichas estructuras, se pueda tener la totalidad de los datos de la empresa a efectos de chequeo y verificación de los totales de una nómina mensual contra los valores de una entidad bancaria. Tópico de estructuras de datos que será desarrollado en el siguiente capítulo.

Ejemplo 3.50 Coeficiente de Correlación

Dada una clase de Algoritmia y Programación, la cual tiene $n$ estudiantes, para cada estudiante se leen dos variables: $X_i$, definida como el coeficiente intelectual, y $Y_i$, identificada como rendimiento académico. Diseñe un algoritmo para calcular el Coeficiente de Correlación $(R_{xy})$, que es un índice que mide la relación lineal entre dos variables cuantitativas y está definido por la expresión

$$R_{xy} = \frac{\frac{\sum_{i=1}^{n} x_i y_i}{n} - \overline{xy}}{S_x S_y}.$$

Los valores de $S_x$ y $S_y$ están definidos por las expresiones

$$S_x = \sqrt{\frac{\sum_{i=1}^{n} x_i x_i}{n} - \overline{x}^2}$$

donde el promedio está definido como

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}.$$

Análogamente,

$$S_y = \sqrt{\frac{\sum_{i=1}^{n} y_i y_i}{n} - \overline{y}^2}, \quad \overline{y} = \frac{\sum_{i=1}^{n} y_i}{n}.$$

*Análisis*: En primer lugar se lee el número de estudiantes del curso ($n$); en segundo lugar se lee para cada estudiante los datos de las variables del coeficiente intelectual ($x_i$) y del rendimiento académico ($y_i$). Seguidamente se va calculando las sumatorias requeridas de los promedios, las dos desviaciones estándar y el coeficiente de correlación; para finalmente construir la expresión matemática del coeficiente.

**Proc**: Coeficiente de correlación

**Entero:** *n*, *i*
**Real:** *sumac*, *sumar*, *sumapc*, *sumapr*, *sumapcr*, *x*, *y*, *promc*, *promc*
**Real:** *promr*, *prom*, *prompc*, *prompr*, *prompcr*, *desvc*, *desvr*, *C*
**Lea** *n* // Leer el número de estudiantes del curso

*sumac* ← 0 // Suma del coeficiente intelectual
*sumar* ← 0 // Suma del rendimiento académico
*sumapc* ← 0 // Suma del producto de coeficiente intelectual
*sumapr* ← 0 // Suma del producto del rendimiento académico
*sumapcr* ← 0 // Suma del producto del coeficiente por
// el rendimiento

// Ciclo repetitivo para calcular las sumatorias del coeficiente
**Para**($i = 1, n, 1$)**Haga**
  **Lea** *x* // Leer el coeficiente intelectual asociado al alumno *i*
  **Lea** *y* // Leer el rendimiento académico asociado al alumno *i*

  $sumac \leftarrow sumac + x$
  $sumar \leftarrow sumar + y$
  $sumapc \leftarrow sumapc + x * x$
  $sumapr \leftarrow sumapr + y * y$
  $sumapcr \leftarrow sumapcr + x * y$
**Fin_Para**

// Cálculo de promedios
$promc \leftarrow sumac/n$
$promr \leftarrow sumar/n$
$prompc \leftarrow sumapc/n$
$prompr \leftarrow sumapr/n$
$prompcr \leftarrow sumapcr/n$

1

1
// Raiz es la función matemática de la raíz cuadrada
$desvc \leftarrow \sqrt{prompc - (promc * promc)}$
$desvr \leftarrow \sqrt{prompr - (promr * promr)}$

// Cálculo del coeficiente $R_{xy}$
$C \leftarrow ((prompcr) - (promc * promr))/(desvc * desvr)$
**Escriba** "El valor del coeficiente de correlación es", $C$
**Fin_proc**

## 3.7 Estructura lógica repetitiva Mientras que

La lógica de control de la estructura repetitiva Mientras que, cuya identidad en el algoritmo se abrevia con la sigla Mq, permite la realización de un conjunto de estructuras lógicas básicas o primitivas lógicas ($p_i$), las cuales se ejecutan cuando la condición del Mientras que es verdadera.

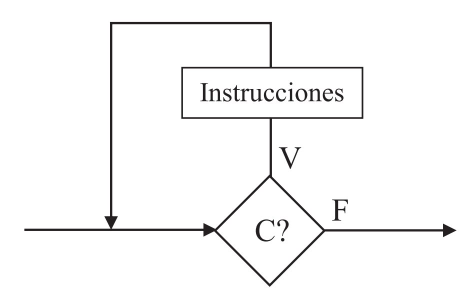



*Figura 3.7* Estructura Repetitiva Mientras que

Implica, por lo tanto, en la estructura Mientras que el cumplimiento cuando sea verdadera de una condición, lo cual en la notación de algoritmos se expresa como:

Dado un conjunto de primitivas $p_1$, $p_2$, $p_3$, ..., $p_i$, ..., $p_n$, la repetitiva Mq se denota en el algoritmo como

**Mientras_que** (condición) **haga**

$p_1, p_2, \ldots, p_n$

**Fin_Mientras_que**

O, en su defecto, también se puede presentar como

**Mq** (condición) **haga**

$p_1, p_2, \ldots, p_n$

**Fin_Mq**

Esta estructura de control de repetición de otras estructuras algorítmicas permite la ejecución de las primitivas $p_1$, $p_2$, $p_3$, ..., $p_n$, siempre y cuando la condición sea verdadera; condición que "... es necesario que no se quede ejecutando indefinidamente..." (Rueda, 1981, p. 21), por cuanto el algoritmo, utilizando la estructura Mientras que, necesariamente debe terminar, a fin de dar los resultados al usuario.

Ejemplo 3.51 Descomposición de un número $k$ en sus dígitos

Dado un número $k$ que pertenece a los enteros positivos, diseñe un algoritmo para descomponer el número en sus dígitos decimales de unidades, decenas, centenas, etc. Si el número es 789, el resultado será 9 (unidades), 8 (decenas) y 7 centenas.

*Análisis*: Se guarda el número original ($k$) en una variable temporal ($k1$). Suponga que el número original es $k = k1 = 789$, este número en el sistema de numeración digital es igual a $7*10^2+8*10^1+9*10^0$; entonces al número le sacamos su residuo utilizando la función Módulo 10 (789 **mod** 10 es 9); el residuo de la aplicación de la función módulo 10 es el dígito posicionado en el lugar de las unidades (1), decenas (2), centenas (3), etc., cuantas veces hayamos aplicado la función módulo sobre la variable temporal ($k1$) ; luego actualizamos la variable temporal, dividiéndola por 10, haciendo el proceso siempre que el cociente sea mayor que cero.

```
Proc: Descomposición de un k en sus dígitos
| Entero: k, ki, j, Residuo
| Lea k // Leer el número a descomponer
| k1 ← k // Se almacena el número a descomponer en una variable
| // temporal
| j ← 0
| // j contiene el exponente de posicionamiento de la descomposición
| Mq(k1 > 0)Haga
| | // Se calcula el residuo, utilizando la función Módulo 10
| | Residuo ← k1 mod 10
| | Escriba Residuo, "*10ˆ", j
| | // El residuo es el dígito ya descompuesto de k
| | k1 ← k1/10 // Se actualiza la variable temporal
| | j ← j + 1 // Actualiza el exponente de posicionamiento del residuo
| Fin_Mq
Fin_proc
```

Cabe resaltar que la división $k1/10$ es entera, dado que $k1$ y 10 son variables enteras. Por ejemplo, si $k1 = 123$, entonces $k1/10 = 123/10 = 12$.

Ejemplo 3.52 Función $\log_2 n$ para potencias de 2

Dado un número $n$, con las características de que sea par, potencia de dos y perteneciente a los enteros positivos, diseñe un algoritmo utilizando la estructura Mientras que para calcular el número de potencia del número $n$.

*Análisis*: Como el número es par y potencia de dos, suponga $n = 8$. Si a este número lo dividimos sucesivamente por 2 hasta 1, encontramos en la división las potencia de $n$, de la siguiente forma: $8/2 = 4$; $4/2 = 2$; $2/2 = 1$; luego, se realizaron tres divisiones; por lo tanto, $2^3 = 8$, con lo cual se concluye que el número de potencias de 8 son tres.

```
Proc: Función log2 n para potencias de 2
| Entero: n, n1, p
| Lea n
| n1 ← n // Guardar el número que se va a descomponer en una variable
| // temporal n1
| p ← 0 // Inicializar el contador de potencia de dos en cero
| Mq(n1 ≠ 1)Haga
| | n1 ← n1/2 // División entera que parte cada vez el número en dos
| | p ← p + 1 // Actualizar el contador de potencias
| Fin_Mq
| Escriba "El logaritmo en base 2 de ", n
| Escriba "Es igual a ", p
Fin_proc
```

Ejemplo 3.53 Impresión de series alternantes de números y sus sumas

Utilizando la estructura de control repetitiva Mientras que (Mq) diseñe un algoritmo para calcular e imprimir los $k$ primeros términos de la serie $S1 = 12-11+24-22+36-33+48-44\ldots$ y adicionalmente los $n$ primeros términos de la serie $S_2 = 10 - 20 + 30 - 40 \ldots$. Escribiendo el resultado de la suma de cada una de las series.

*Análisis*: La primera serie corresponde a una sucesión alternante (+ y −) de los múltiplos de 12 y 11; por otra parte, la segunda sucesión también es alternante pero de los múltiplos de 10.

```
Proc: Impresión de series oscilantes
| Entero: k, S1, S2, i, j, T1, T2, c
| Lea k
| S1 ← 0 // Inicializa la primera serie
| S2 ← 0 // Inicializa la segunda serie
| // Estructura repetitiva que permite el cálculo de la primera serie
1
```

```
1
| i ← 1 // Controla el ciclo Mq con el número de términos de la serie
| j ← 1 // Coeficiente base para calcular cada término
| Mq(i ≤ k)Haga
| | T1 ← j * 12 // Calcula el término positivo
| | T2 ← j * 11 // Calcula el término negativo
| | S1 ← S1 + T1 − T2 // Se le suma los términos calculados
| | j ← j + 1 // Coeficiente base para el otro término
| | i ← i + 2 // Se incrementa i para alcanzar el valor k
| Fin_Mq
| // Vea que el ciclo anterior asigna de 2 en 2
| // A continuación se tiene en cuenta si k es un número impar
| Si(k mod 2 = 1)Entonces
| | T1 ← c * j * 12
| | S1 ← S1 + T1
| F.Si
|
| j ← 1 // Controla el segundo Mq, que calcula la serie dos
| c ← 1 // Signo del coeficiente del término de la serie
| Mq(j ≤ k)Haga
| | T ← c * j * 10 // Calcula el término de la segunda serie
| | S2 ← S2 + T // Calcula la suma de la serie dos
| | c ← c * (−1) // Varia el signo como base para calcular otro término
| | j ← j + 1 // Actualiza el contador de términos para alcanzar n
| Fin_Mq
| Escriba "El valor de las serie alternante uno es ", S1
| Escriba "El valor de las serie alternante uno es ", S2
Fin_proc
```

---

Ejemplo 3.54 Multiplicación rusa

Dados dos números $n$ y $m$, identificados como multiplicando ($n$) y multiplicador ($m$), el algoritmo de la multiplicación rusa divide por dos el multiplicador hasta que su valor es 1 y multiplica por dos el multiplicando; adicionalmente, para obtener el resultado de la multiplicación suma todos los multiplicandos correspondientes a los multiplicadores impares, y esta suma es el resultado de la multiplicación. Diseñe un algoritmo para multiplicar dos números dados utilizando la multiplicación rusa.

*Análisis*: Sea $n = 12$ el multiplicando; y sea $m = 10$ el multiplicador. Entonces, la multiplicación normal da como resultado $12 \times 10 = 120$ y la multiplicación rusa utiliza el siguiente procedimiento:

| Multiplicando | | | Multiplicador | | |
|---|---|---|---|---|---|
| $n$ | $=$ | 12 | $m$ | $=$ | 10 |
| $12 \times 2$ | $=$ | **24** | $10/2$ | $=$ | **5** |
| $24 \times 2$ | $=$ | 48 | $5/2$ | $=$ | 2 (se utiliza la división entera) |
| $48 \times 2$ | $=$ | **96** | $2/2$ | $=$ | **1** |

El resultado de la multiplicación es entonces $24 + 96 = 120$ (porque son los multiplicandos de los multiplicadores que dan resultados impares).

```
Proc: Multiplicación rusa
  Entero: n, m, Suma, Mul, Dor
  Lea n, m
  Suma ← 0 // Acumula el valor de la multiplicación rusa
  // Estructura repetitiva que permite el control de las multiplicaciones
  // y las divisiones al igual que va sumando los multiplicandos que
  // corresponden a los multiplicadores impares
  Mul ← n // Variable temporal correspondientes al
  // Multiplicando y sus valores
  Dor ← m // Variable temporal que almacena el multiplicador
  // y sus valores
  Mq(Dor ≥ 1)Haga
    // Comprueba si el multiplicador es impar
    Si(Dor mod 2 = 1)Entonces
      // Adiciona el multiplicando por 2 al resultado
      Suma ← Suma + Mul
    F.Si
    Mul ← Mul * 2
    Dor ← Dor/2
  Fin_Mq
  Escriba “El valor de la multiplicación rusa de ”, n
  Escriba “por ”, m, “ es ”, Suma
Fin_proc
```

Ejemplo 3.55 Juego: Adivinar números entre dos jugadores: A y B

Dados dos jugadores, A y B, suponga que el jugador A tiene $n$ números cuyos rangos están ubicados en el intervalo de los enteros positivos de $[1, k]$. El jugador B debe adivinar el número que el jugador A tiene en mente. Diseñe un algoritmo que lea los $n$ números del jugador A y dé la posibilidad al jugador B de dar un número, el cual es leído hasta que coincida con el que tiene el jugador A. El algoritmo debe contar el número de veces en que el jugador B utilizó para adivinar el número del jugador A, al igual que calcular el promedio del número de veces que empleó el jugador B en adivinar todos los números del jugador A.

```
Proc: Juego de adivinanzas
| Entero: n, k, i, Sp, nA, sw, cont, nB
| Real: p
| Lea n, k
| i ← 1 // Indice que permite controlar el número de números
| // propuestos por A
| Sp ← 0 // Variable que se va a utilizar para sacar el promedio
| Mq(i ≤ n)Haga
| | Lea nA // Número dado por el jugador A
| | Si((nA ≥ 1) y (nA ≤ k))Entonces
| | | sw ← 0 // Controla si el número B adivina el número de A
| | | cont ← 0 // Intentos que emplea B en adivinar el número
| | | Mq(sw = 0)Haga
| | | | Lea nB // Número de B
| | | | cont ← cont + 1
| | | | Si(nA = nB)Entonces
| | | | | // El jugador B ha adivinado el número de A
| | | | | sw ← 1 // Se varía el switch de coincidencia
| | | | | // Esta variación obliga a que se cierre el ciclo Mq interno
| | | | F.Si
| | | | Escriba "Intentos: ", cont
| | | | Sp ← Sp + cont
| | | | i ← i + 1
| | | Fin_Mq
| | Sino
| | | Escriba "El número de A no está en el intervalo [1,", k, "]"
| | F.Si
| Fin_Mq
| // Cálculo del promedio
| p ← Sp/n
| Escriba "El número promedio de intentos que empleó B para adivinar"
| Escriba "los n números de A es igual a ", p
Fin_proc
```

## 3.8 Estructura lógica repetitiva Haga Hasta

La estructura de control repetitiva Haga Hasta permite la ejecución de un conjunto de estructuras de control o primitivas $p_1$, $p_2$, $p_3$, ..., $p_i$, ..., $p_n$, hasta que se cumpla la condición explicitada en la clausula de control del Haga Hasta.

Las primitivas de control contenidas dentro del ciclo repetitivo Haga Hasta se ejecutan hasta que la condición sea verdadera. Lo anterior implica que en esta estructura como mínimo las primitivas de control $p_1$, $p_2$, $p_3$, ..., $p_i$, ..., $p_n$, se ejecutan

al menos una vez.

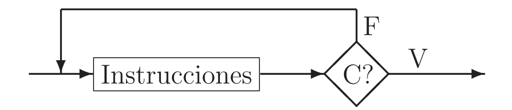



*Figura 3.8* Estructura Repetitiva Haga Hasta

El ciclo repetitivo Haga Hasta se abrevia con HH, y su estructura en el algoritmo se denota como

**Haga_Hasta**

$p_1, p_2, p_3, \ldots, p_n$

**Fin_Haga_Hasta**(Condición)

Estructura de control que es equivalente en su notación de algoritmos a

**HH**

$p_1, p_2, p_3, \ldots, p_n$

**Fin_HH**(Condición)

Las estructuras de control repetitivas Mientras que y Haga Hasta expuestas permiten la ejecución de un conjunto de primitivas algorítmicas; pero en las dos estructuras relacionadas (Mq) y (HH) hay una diferencia fundamental: en la estructura Mientras que la condición de verdad para ejecutar las instrucciones que van en el interior se evalúa antes de ejecutar las primitivas; luego, es posible que el conjunto de primitivas en el interior del ciclo (Mq) no se ejecute ni una sola vez si la condición no se cumple. En el caso de la primitiva Haga Hasta, las instrucciones contenidas dentro del ciclo repetitivo (HH) se ejecutan al menos una vez, y después sí se evalúa la condición que controla la repetición del Haga Hasta.

Ejemplo 3.56 Verificador digital

Un sistema de codificación digital considera como códigos válidos los situados en el intervalo cerrado $[10'000\,000, 99'999\,999]$; los códigos ubicados en rangos diferentes al intervalo dado son considerados errados. Diseñe un algoritmo que haga el proceso de verificación de contar e imprimir códigos válidos, terminando el proceso cuando ingresen un código ubicado en un rango diferente al rango dado. (Los algoritmos de verificación son útiles para garantizar la entrada de NIT (cédulas) a un sistema de información).

*Análisis*: Suponga los códigos de entrada $10'000\,000$, $50'500\,500$, $99'784\,777$, 9999, entonces los tres primeros códigos dados son válidos, y en tal sentido, el sistema verificador capturaría solo tres códigos válidos. Ahora, el código 9999 no es válido porque su valor es inferior a $10'000\,000$.

**Proc**: Verificador digital
**Entero:** *cont*, *cod*
*cont* ← 0 // Cuenta el número de códigos digitales válidos en el rango
// dado en el intervalo $[10'000\,000, 99'999\,999]$
// Ciclo repetitivo que lee, cuenta e imprime los códigos
**HH**
**Lea** *cod* // Código que se debe verificar si está en el rango dado
**Si**((($cod \geq 10000000$) y ($cod \leq 99999999$))**Entonces**
**Escriba** "Código válido"
*cont* ← *cont* + 1
**Sino**
**Escriba** "El código digitado NO es válido"
**Escriba** "Fin del proceso de verificación del código ", *cod*
**F.Si**
**Fin_HH**((($cod < 10000000$) o ($cod > 99999999$))
**Escriba** "Número de códigos válidos = ", *cont*
**Fin_proc**

---

Ejemplo 3.57 Controlador de tanques

---

Un sistema de tanques para riego está compuesto de un tanque principal ($Tp$) y tres tanques secundarios ($T1, T2$, y $T3$). El tanque principal se llena a través de una tubería madre con un flujo de entrada de $k$ litros por segundo, y a su vez alimenta a los tanques secundarios dispuestos en secuencia para su llenado en el orden $T1 \rightarrow T2 \rightarrow T3$. La velocidad de entrada del tanque principal al tanque uno es de $m$ litros de agua por segundo, del tanque uno al tanque dos es de $n$ litros por segundo y del tanque dos al tres de $p$ litros por segundo. Los tanques secundarios tienen tres flujos de salida: el primero, un flujo de riego de $z$ litros por segundo; el segundo, un flujo de riego de a litros por segundo; y el tercero un caudal de riego de $t$ litros/segundo. Los flujos de riego y alimentación de los tanques secundarios operan únicamente hasta que cada uno de ellos llegue a un determinado nivel de reserva. Los niveles de reserva de los tres tanques son: nivel de reserva del tanque uno ($N1$), nivel de reserva del tanque dos ($N2$) y nivel de reserva del tanque tres ($N3$); niveles de reserva que deben ser leídos como datos de entrada. Teniendo en cuenta que $k \gg m \gg n \gg p \gg z \gg a \gg t$, y que cada uno de los tanques tiene un control que identifica el tope o nivel límite del tanque, con el fin de que el agua no se desperdicie, diseñe un algoritmo que controle el sistema de tanques.

**Proc**: Controlador de tanques
**Real:** $N1$, $N2$, $N3$, $k$, $m$, $n$, $p$, $z$, $a$, $t$, $Vp$, $V1$, $V2$, $V3$
**Entero:** $SwTp$, $Sw1$, $Sw2$, $Sw3$
**Lea** $N1, N2, N3$ // Leer los niveles de reserva de los tanques
**Lea** $k, m, n, p$ // Leer los flujos de entrada de los tanque en
1

```
1
// litros por segundo
Lea z, a, t  // Caudal de riego de los tanques T1(z), T2(a) y T3(t)
// Volúmenes de tanques principal y secundarios vacíos
Vp ← 0, V1 ← 0, V2 ← 0, V3 ← 0
SwTp ← 0  // Switch que controla el llenado del tanque principal
// Controladores del llenado de los tres tanques
Sw1 ← 0, Sw2 ← 0, Sw3 ← 0
// Ciclo repetitivo que controla el nivel del tanque principal
HH
  Vp ← Vp + k  // Volumen del tanque principal con carga de entrada k
  HH
    Vp ← Vp − m  // Reducción del volumen del tanque principal
    V1 ← V1 + m  // Volumen del tanque secundario uno
    Si(V1 > N1 + z)Entonces
      V1 ← V1 − z  // Volumen de tanque mayor que nivel de reserva
      HH
        V1 ← V1 − n
        V2 ← V2 + n
        Si(V2 > N2 + a)Entonces
          V2 ← V2 − a
          HH
            V2 ← V2 − p
            V3 ← V3 + p
            Si(V3 > N3 + t)Entonces
              V3 ← V3 − t
            Sino
              Escriba "Tanque 3 sin reserva"
            F.Si
            Lea Sw3
          Fin_HH(Sw3 = 1)
        Sino
          Escriba "Tanque 2 sin reserva"
        F.Si
        Lea Sw2
      Fin_HH(Sw2 = 1)
    Sino
      Escriba "Tanque 1 sin reserva"
    F.Si
    Lea Sw1
  Fin_HH(Sw1 = 1)
  Lea SwTp
Fin_HH(SwTp = 1)
Fin_proc
```

Ejemplo 3.58 Evaluador bancario de empleados

Un sistema bancario atiende a sus clientes a nivel personalizado a través de tres empleados que desempeñan las funciones de cajeros manuales. El banco está interesado en conocer el nivel de atención de sus empleados para con los clientes, para lo cual requiere la siguiente información:

*i*) Número total de clientes atendidos por el banco durante una jornada horaria.

*ii*) Número total de clientes atendidos por cada cajero.

*iii*) Monto total de transacciones por cada cajero.

*iv*) Promedio de transacciones de cada cajero.

*v*) El promedio de transacciones del banco.

*vi*) El cajero que realizó el mayor número de transacciones y su monto.

*vii*) El cajero que realizó el menor número de transacciones y su monto.

Teniendo en cuenta que *i*) un cliente puede realizar más de una transacción ante el cajero; *ii*) el cliente puede decidir pasar de un cajero a otro, con la condición de que solo consulta cada cajero manual una sola vez; *iii*) las operaciones bancarias se realizan hasta que la entidad bancaria se encuentre abierta. Diseñe un algoritmo representativo del evaluador bancario.

*Análisis*: El algoritmo de control del sistema de evaluación de empleados del sector bancario requiere tres procesos principales: *i*) el proceso de inicialización de variables; *ii*) el proceso de control bancario de atención a los clientes; *iii*) el proceso de arqueo o consolidación estadística de las transacciones bancarias.

En el proceso de inicialización de variables se colocan en cero los valores representativos de las estadísticas bancarias y los *flags* o *switchs* que apoyan el proceso del control del banco. La inicialización de las estadísticas bancarias requeridas son: los contadores de clientes atendidos por cada cajero ($c1$, $c2$ y $c3$), el monto de las transacciones recibidas por cada cajero ($m1$, $m2$ y $m3$), los contadores del número de transacciones por cajero ($ct1$, $ct2$ y $ct3$) y la suma base para el promedio de transacciones totales del banco $st$. Adicionalmente se requiere el switch que controla la apertura del banco ($SwA$) y el switch que controla las transacciones de los clientes ($SwT$).

El proceso de control bancario, previamente habiendo inicializado los switches mencionados, es controlado algorítmicamente por dos estructuras de control: la primera un Haga Hasta (HH) con control de finalización del estado de apertura del banco ($SwA$), que en el caso de estar abierta la entidad lee las variables que controlan los cajeros del tipos uno ($t1$), dos ($t2$) y tres ($t3$); la segunda también es una estructura de control Haga Hasta (HH) que en función del valor del switch de transacciones del cliente ($SwT$) sigue o no

realizando transacciones; la última estructura (HH) se encarga de acumular las estadísticas requeridas para cada cajero en términos del monto y del número de transacciones por cajero.

Finalmente, el arqueo estadístico por cajero y el arqueo total del banco son instrucciones de control de sumas y promedios particulares a cada cajero y al banco en los totales de transacciones diarias.

```
Proc: Evaluador bancario de empleados
| Entero: SwA, c1, c2, c3, m1, m2, m3, ct1, ct2, ct3, st, cc, tc, t1
| Entero: SwT, mtc, t2, t3, Mayor, Mmayor, Menor, Mmenor
| Real: ptc1, ptc2, ptc3, ptb
| // Switch que controla si el banco está abierto
| SwA ← 1
| // Contadores de cliente atendidos por cada cajero
| c1 ← 0, c2 ← 0, c3 ← 0
| // Monto de transacciones realizadas por cajero
| m1 ← 0, m2 ← 0, m3 ← 0
| // Contador de transacciones por cajero
| ct1 ← 0, ct2 ← 0, ct3 ← 0
| st ← 0  // Base para el promedio de totales del banco
| cc ← 0  // Contador de clientes totales atendidos
| tc ← 0  // Total de clientes atendidos en una jornada
|
| HH
| | Lea t1  // Variable que controla el cajero C1 que atiende al cliente
| | Si(t1 = 1)Entonces
| | | SwT ← 1  // Controla las transacciones del cliente
| | | c1 ← c1 + 1  // Cliente atendidos por el cajero uno
| | | HH
| | | | Lea mtc  // Monto de la transacción del cliente
| | | | m1 ← m1 + mtc  // Monto total del cajero uno
| | | | st ← st + mtc  // Suma total de transacciones
| | | | ct1 ← ct1 + 1  // Actualiza el total de transacciones
| | | | Lea SwT
| | | Fin_HH(SwT = 0)
| | F.Si
| | Lea t2  // Variable que controla el cajero C2 que atiende al cliente
| | Si(t2 = 2)Entonces
| | | SwT ← 1
| | | c2 ← c2 + 1  // Cliente atendidos por el cajero dos
| | | HH
| | | | Lea mtc
| | | | m2 ← m2 + mtc
1 2 3 4
```

```
1 2 3 4
      st ← st + mtc
      ct2 ← ct2 + 1
      Lea SwT
    Fin_HH(SwT = 0)
  F.Si
  // Variable que controla el cajero C3 que atiende al cliente
  Lea t3
  Si(t3 = 3)Entonces
    SwT ← 1
    c3 ← c3 + 1
    HH
      Lea mtc
      m3 ← m3 + mtc
      st ← st + mtc
      ct3 ← ct3 + 1
      Lea SwT
    Fin_HH(SwT = 0)
  F.Si
  Lea SwA
Fin_HH(SwA = 0)
// Cálculo de totales
// Total de clientes atendidos en el banco en una jornada
tc ← tc + c1 + c2 + c3
Escriba "El total de clientes atendidos en una jornada diaria es ", tc
// Número total de clientes atendidos por cada cajero
Escriba "El número total de cliente atendidos por C1 es ", c1
Escriba "El número total de cliente atendidos por C2 es ", c2
Escriba "El número total de cliente atendidos por C3 es ", c3
// Monto total de transacciones por cada cajero
Escriba "El monto total de transacciones realizadas por C1 es ", m1
Escriba "El monto total de transacciones realizadas por C2 es ", m2
Escriba "El monto total de transacciones realizadas por C3 es ", m3
// Promedio de transacciones de cada cajero
ptc1 ← m1/ct1, ptc2 ← m2/ct2, ptc3 ← m3/ct3
Escriba "El promedio de transacciones realizadas por C1 es ", ptc1
Escriba "El promedio de transacciones realizadas por C2 es ", ptc2
Escriba "El promedio de transacciones realizadas por C3 es ", ptc3
// El promedio de transacciones del banco
ptb ← st/(ct1 + ct2 + ct3)
Escriba "El promedio diario de transacciones del banco es ", ptb
// Ahora se escoge el cajero que realizó el mayor número de
// transacciones y su monto correspondiente
Mayor ← ct1
1
```

```
1
| Mmayor ← m1
| Si(ct2 > Mayor)Entonces
| | Mayor ← ct2
| | Mmayor ← m2
| F.Si
| Si(ct3 > Mayor)Entonces
| | Mayor ← ct3
| | Mmayor ← m3
| F.Si
| // Ahora se escoge el cajero que realizó el menor número de
| // transacciones y su monto correspondiente
| Menor ← ct1
| Mmenor ← m1
| Si(ct2 < Menor)Entonces
| | Menor ← ct2
| | Mmenor ← m2
| F.Si
| Si(ct3 < Menor)Entonces
| | Menor ← ct3
| | Mmenor ← m3
| F.Si
| Escriba "Cajero con el mayor de transacciones = ", Mayor
| Escriba "Monto del cajero con el mayor número de transacciones = "
| Escriba Mmayor
| Escriba "Cajero con el menor de transacciones = ", Menor
| Escriba "Monto del cajero con el menor número de transacciones = "
| Escriba Mmenor
Fin_proc
```

Ejemplo 3.59 Controlador de entradas de los alumnos en un Sistema Universitario

Un sistema universitario tiene para el registro de entradas de los estudiantes tres torniquetes. Diseñe un algoritmo que cuente el número de estudiantes diarios que pasan por cada torniquete, llevando una cuenta semanal (7 días de la semana) de los alumnos, como también el número de alumnos por programa que pasan al día por cada torniquete de los programas de la División de Ingeniería: Ingeniería Civil ($p1$), Eléctrica ($p2$), Electrónica ($p3$), Industrial ($p4$), Mecánica ($p5$) e Ingeniería de Sistemas ($p6$).

*Análisis*: El controlador de entradas de estudiantes para un sistema universitario se fundamenta en tres procesos:

I) La inicialización de las variables que representan las entradas diarias de los alumnos por los torniquetes ($t1$, $t2$ y $t3$) y para los seis programas

y los tres torniquetes (($p1t1$, $p2t1$, $p3t1$, $p4t1$, $p5t1$, $p6t1$), ($p1t2$, $p2t2$, $p3t2$, $p4t2$, $p5t2$, $p6t2$), ($p1t3$, $p2t3$, $p3t3$, $p4t4$, $p5t5$)). Adicionalmente se requiere el control de los siete días de la semana.

ii) El proceso de control semanal de los tres torniquetes se puede realizar o con una estructura de lógica de control Para o con una estructura de control Mientras que (Mq); dentro de las estructuras antes mencionadas se lee el *switch* de apertura de los torniquetes ($SwA$), y con un controlador Haga Hasta (HH) se distribuyen los estudiantes que pasan diariamente por cada torniquete, leyendo en el interior del HH tanto el número del torniquete ($n$) como el código del alumno ($c$).

iii) Finalmente, antes de avanzar al siguiente día de la semana se totalizan los acumuladores semanales de los torniquetes ($t1s$, $t2s$, y $t3s$).

```
Proc: Controlador de entradas en un sistema universitario
| Entero: t1, t2, t3, t1s, t2s, t3s, p1t1, p2t1, p3t1, p4t1, p5t1, p6t1, p1t2
| Entero: p2t2, p3t2, p4t2, p5t2, p6t2, p1t3, p2t3, p3t3, p4t3, p5t3, p6t3
| Entero: i, SwA, n, c
| // Contadores de torniquetes (por día y semana)
| t1 ← 0; t2 ← 0; t3 ← 0
| t1s ← 0; t2s ← 0; t3s ← 0
| // Contadores diarios por programa por torniquete
| p1t1 ← 0; p2t1 ← 0; p3t1 ← 0; p4t1 ← 0; p5t1 ← 0; p6t1 ← 0
| p1t2 ← 0; p2t2 ← 0; p3t2 ← 0; p4t2 ← 0; p5t2 ← 0; p6t2 ← 0
| p1t3 ← 0; p2t3 ← 0; p3t3 ← 0; p4t3 ← 0; p5t3 ← 0; p6t3 ← 0
| i ← 1 // Control semanal
| Mq(i ≤ 7)Haga
| | SwA ← 1 // Switch de apertura de los torniquetes
| | HH
| | | Lea n, c
| | | DD(n)Haga
| | | | Opción 1 :
| | | | | // Se aumenta el contador diario de primer torniquete
| | | | | t1 ← t1 + 1
| | | | | DD(c)Haga
| | | | | | Opción 1 :
| | | | | | | p1t1 ← p1t1 + 1
| | | | | | Fin
| | | | | | Opción 2 :
| | | | | | | p2t1 ← p2t1 + 1
| | | | | | Fin
| | | | | | Opción 3 :
| | | | | | | p3t1 ← p3t1 + 1
| | | | | | Fin
1 2 3 4 5 6
```

```
1 2 3 4 5 6
          Opción 4 :
            p4t1 ← p4t1 + 1
          Fin
          Opción 5 :
            p5t1 ← p5t1 + 1
          Fin
          Opción 6 :
            p6t1 ← p6t1 + 1
          Fin
        Fin_DD
      Fin
      Opción 2 :
        // Se aumenta el contador diario del segundo torniquete
        t2 ← t2 + 1
        DD(c)Haga
          Opción 1 :
            p1t2 ← p1t2 + 1
          Fin
          Opción 2 :
            p2t2 ← p2t2 + 1
          Fin
          Opción 3 :
            p3t2 ← p3t2 + 1
          Fin
          Opción 4 :
            p4t2 ← p4t2 + 1
          Fin
          Opción 5 :
            p5t2 ← p5t2 + 1
          Fin
          Opción 6 :
            p6t2 ← p6t2 + 1
          Fin
        Fin_DD
      Fin
      Sino :
        // Se aumenta el contador diario del tercer torniquete
        t3 ← t3 + 1
        DD(c)Haga
          Opción 1 :
            p1t3 ← p1t3 + 1
          Fin
1 2 3 4 5 6
```

```
1 2 3 4 5 6
          Opción 2 :
            p2t3 ← p2t3 + 1
          Fin
          Opción 3 :
            p3t3 ← p3t3 + 1
          Fin
          Opción 4 :
            p4t3 ← p4t3 + 1
          Fin
          Opción 5 :
            p5t3 ← p5t3 + 1
          Fin
          Opción 6 :
            p6t3 ← p6t3 + 1
          Fin
        Fin_DD
      Fin
    Fin_DD
    Lea SwA
  Fin_HH(SwA = 0)
  // Acumulador semanal
  t1s ← t1s + t1; t2s ← t2s + t2; t3s ← t3s + t3;
  Escriba "Número de estudiantes por torniquete al día ", t1,t2,t3
  Escriba "Est. por programa en el torniquete 1 "
  Escriba p1t1,p2t1,p3t1,p4t1,p5t1,p6t1
  Escriba "Est. por programa en el torniquete 2 "
  Escriba p1t2,p2t2,p3t2,p4t2,p5t2,p6t2
  Escriba "Est. por programa en el torniquete 3 "
  Escriba p1t3,p2t3,p3t3,p4t3,p5t3,p6t3
  p1t1 ← 0; p2t1 ← p3t1; p4t1 ← 0; p5t1 ← 0; p6t1 ← 0
  p1t2 ← 0; p2t2 ← 0p3t2; p4t2 ← 0; p5t2 ← 0; p6t2 ← 0
  p1t3 ← 0; p2t3 ← 0p3t3; p4t3 ← 0; p5t3 ← 0; p6t3 ← 0
  i ← i + 1
 Fin_Mq
 Escriba "El número de alumnos que ingresaron por semana es "
 Escriba t1s,t2s,t3s
Fin_proc
```

---

Ejemplo 3.60 Administrador de edificios

Un administrador gestiona $k$ edificios; en cada uno de los edificios hay $n$ apartamentos y cada apartamento paga un costo $c$ de administración; pago que se efectúa si el bit (o switch) de pago viene en 1. Diseñe un algoritmo que utilizando un HH para controlar el número de edificios y un Mq para

controlar los apartamentos de los edificios genere un informe del valor total pagado en cada edificio, el número de apartamentos que pagaron en cada edificio, el número de apartamentos que no pagaron en cada edificio en un determinado mes, y finalmente el valor recaudado por todos los edificios en el mes mencionado.

*Análisis*: Suponga que el administrador hace la gestión de cuatro edificios, mostrados gráficamente en la tabla 3.5.

El edificio 4 tiene un total de 15 apartamentos; los apartamentos numerados del 1 al 9 del edificio 4 no pagaron la cuota mensual; por lo tanto, el bit de pago ($p$) está en cero y, consecuentemente, no hay una cuota de administración asociada ($c$); los apartamentos numerados en el rango $[10, 15]$ del mismo edificio sí pagaron, y hay para cada uno de ellos una cuota de administración asociada que para el total del edificio es $6c$. Finalmente, con base en los recaudos de cada edificio se debe sacar al mes el total de recaudos de cada edificio, que para el caso gráfico presentado son $Recaudos = 20 * c + 4 * c + 1 * c + 6 * c$.

*Tabla 3.5* Gestor de edificios

<table>
<tr><td>20</td><td rowspan="20">$p = 1; c$</td><td colspan="6" rowspan="5"></td></tr>
<tr><td>19</td></tr>
<tr><td>18</td></tr>
<tr><td>17</td></tr>
<tr><td>16</td></tr>
<tr><td>15</td><td colspan="4" rowspan="5"></td><td>15</td><td rowspan="6">$p = 1; c$</td></tr>
<tr><td>14</td><td>14</td></tr>
<tr><td>13</td><td>13</td></tr>
<tr><td>12</td><td>12</td></tr>
<tr><td>11</td><td>11</td></tr>
<tr><td>10</td><td colspan="2" rowspan="3"></td><td>10</td><td rowspan="6">$p = 0$</td><td>10</td></tr>
<tr><td>9</td><td>9</td><td>9</td><td rowspan="9">$p = 0$</td></tr>
<tr><td>8</td><td>8</td><td>8</td></tr>
<tr><td>7</td><td>7</td><td>$p = 1; c$</td><td>7</td><td>7</td></tr>
<tr><td>6</td><td>6</td><td>$p = 0$</td><td>6</td><td>6</td></tr>
<tr><td>5</td><td>5</td><td>$p = 1; c$</td><td>5</td><td>5</td></tr>
<tr><td>4</td><td>4</td><td>$p = 0$</td><td>4</td><td>$p = 1; c$</td><td>4</td></tr>
<tr><td>3</td><td>3</td><td>$p = 1; c$</td><td>3</td><td rowspan="3">$p = 0$</td><td>3</td></tr>
<tr><td>2</td><td>2</td><td>$p = 0$</td><td>2</td><td>2</td></tr>
<tr><td>1</td><td>1</td><td>$p = 1; c$</td><td>1</td><td>1</td></tr>
<tr><td>Aptos.</td><td>Edificio 1</td><td>Aptos.</td><td>Edificio 2</td><td>Aptos.</td><td>Edificio 3</td><td>Aptos.</td><td>Edificio 4</td></tr>
<tr><td colspan="2">$k = 4$</td><td colspan="2">$k = 3$</td><td colspan="2">$k = 2$</td><td colspan="2">$k = 1$</td></tr>
</table>

**Proc**: Administrador de edificios

**Entero:** $k$, $i$, $Recaudos$, $n$, $Suma$, $Np$, $Nop$, $j$, $switchp$, $c$

**Lea** $k$ // Número de edificios

$i \leftarrow k$ // Controla el número de edificios procesados

$Recaudos \leftarrow 0$ // Acumulador de recaudos de todos los

// edificios en el mes

1

```
1
HH
    Lea n // Número de apartamentos del edificio, asociado k
    Suma ← 0 // Suma de los aportes de administración de todos los
    // apartamentos del edificio i
    Np ← 0 // Contador de los apartamentos que pagaron
    Nop ← 0 // Contador de los apartamentos que no pagaron
    j ← 1
    Mq(j ≤ n)Haga
        Lea switchp // switch de pago de los apartamentos
        Si(switchp = 1)Entonces
            Lea c // Lee la cuota de administración
            Suma ← Suma + c
            Np ← Np + 1
        Sino
            Nop ← Nop + 1
        F.Si
        j ← j + 1
    Fin_Mq
    Escriba "El valor recaudado del edificio ", k, " es igual a ", Suma
    Escriba "Apartamentos que pagaron en el edificio ", k
    Escriba " es igual a ", Np
    Escriba "Apartamentos que no pagaron en el edificio ", k
    Escriba " es igual a ", Nop
    // Suma los recaudos del apartamento k
    Recaudos ← Recaudos + Suma
    k ← k − 1
Fin_HH(k = 0)
Escriba "El valor recaudado durante el mes de los ", k, " edificios "
Escriba "es igual a: ", Recaudos
Fin_proc
```

## 3.9 Algoritmos resueltos

Ejemplo 3.61 Comparación de dos números

Escriba un algoritmo que compare dos números y muestre el mayor.

*Análisis*: Para saber cuál de los dos números $a$ y $b$ leídos es mayor primero se pregunta si son diferentes; luego se compara si $a$ es mayor, y si no concluir que lo es $b$; también se muestra el mensaje en caso de ser iguales. En la evaluación de cada condición se indica si es mayor, menor o igual, respectivamente.

$$3, 6 \implies \boxed{\mathbf{A}} \implies \text{El mayor es el 6}$$

**Proc**: Comparar dos números
  **Entero:** $a$, $b$
  **Escriba** "Inserta dos enteros $a$ y $b$"
  **Lea** $a, b$
  **Si**($a \neq b$)**Entonces**
    **Si**($a > b$)**Entonces**
      **Escriba** $a$, "es mayor que ", $b$
    **Sino**
      **Escriba** $a$, " es menor que ", $b$
    **F.Si**
  **Sino**
    **Escriba** $a$, "es igual a ", $b$
  **F.Si**
**Fin_proc**

---

Ejemplo 3.62 Mayor de un conjunto de números

Escriba un algoritmo que determine el mayor de tres números. En el desarrollo de este algoritmo se proponen dos soluciones.

*Análisis*: Se declaran las variables $x, y, z$ como reales. Se forman condicionales compuestos para mayor claridad y un anidamiento simple.

El orden está en si una variable es la mayor, y en caso de que no lo sea, se pregunta por la siguiente variable, y así sucesivamente. Como los números reales resultan ser un conjunto ordenado, si se preguntó por los $n-1$ primeros números y ninguno resultó ser el mayor, entonces el $n$-ésimo número es el mayor.

Sin embargo, para encontrar el máximo para una cantidad mayor de números resulta ineficiente agregar un sinnúmero de condiciones (Solución 1); en este caso se asigna primeramente a una variable $max$ un número $x_1$, y si para algún $x_k$ se tiene que $x_k > max$, entonces a $max$ se le asigna $x_k$, es decir, $max \leftarrow x_k$. De igual manera se hace para encontrar el mínimo; en este caso la condición sería $x_k < min$. Se hace esto para cada número, realizando un máximo de $n-1$ iteraciones. Usando este método el algoritmo queda como en la Solución 2.

**Solución 1:**
**Proc**: Mayor de 3 números
  **Entero:** $x$, $y$, $z$, $mayor$
1

```
1
| Escriba "Ingrese los tres números: "
| Lea x, y, z
| Si(x > y ∧ x > z)Entonces
| | Escriba "El mayor de los tres números es: ", x
| Sino
| | Si(y > x ∧ y > z)Entonces
| | | Escriba "El mayor de los tres números es: ", y
| | Sino
| | | Escriba "El mayor de los tres números es: ", z
| | F.Si
| F.Si
Fin_proc
```

**Solución 2:**

```
Proc: Mayor de 3 números
| Entero: x, y, z, mayor
| Escriba "Ingrese los tres números: "
| Lea x, y, z
| mayor ← x
| Si(y > mayor)Entonces
| | mayor ← y
| F.Si
| Si(z > mayor)Entonces
| | mayor ← z
| F.Si
| Escriba "El mayor de los tres números es: ", mayor
Fin_proc
```

Ejemplo 3.63 Calificaciones de un estudiante

La valoración que representa las calificaciones de un estudiante se calcula de acuerdo con la siguiente tabla:

| **Calificación numérica** | **Valoración** |
|---|---|
| mayor o igual que 90 | Excelente |
| menor que 90 pero mayor que o igual a 80 | Sobresaliente |
| menor que 80 pero mayor que o igual a 60 | Aceptable |
| menor que 60 | Insuficiente |

*Análisis*: Por ejemplo, si un estudiante tiene una calificación de 54, su valoración será "Insuficiente". Análogamente al anterior ejercicio, se pregunta en un orden, en este caso en el orden de la tabla expuesta. Este ejercicio evita condiciones complejas.

```
Proc: Valoración de un estudiante
| Entero: num
| Escriba "Calificacion numérica: "
| Lea num
| Si(num ≥ 90)Entonces
| | Escriba "La calificación corresponde a Excelente"
| Sino
| | Si(num ≥ 80)Entonces
| | | Escriba "La calificación corresponde a Sobresaliente"
| | Sino
| | | Si(num ≥ 60)Entonces
| | | | Escriba "La calificación corresponde a Aceptable"
| | | Sino
| | | | Escriba "La calificación corresponde a Insuficiente"
| | | F.Si
| | F.Si
| F.Si
Fin_proc
```

Ejemplo 3.64 Conversión de Si-Sino a Dependiendo De

Vuelva a escribir la siguiente cadena Si-Sino usando la instrucción Dependiendo De:

```
Proc: Calificaciones
| Caracter: calif
| Escriba "Calificación en letra: "
| Lea calif
| Si(calif = A)Entonces
| | Escriba "La calificación numérica está entre 90 y 100"
| Sino
| | Si(calif = B)Entonces
| | | Escriba "La calificación numérica está entre 80 y 89.9"
| | Sino
| | | Si(calif = C)Entonces
| | | | Escriba "La calificación numérica está entre 70 y 79.9"
| | | Sino
| | | | Si(calif = D)Entonces
| | | | | Escriba "Mala calificación"
| | | | Sino
| | | | | Escriba "Por supuesto que no tuve nada que ver con mi calificación."
| | | | F.Si
| | | F.Si
| | F.Si
| F.Si
Fin_proc
```

*Análisis*: Se observa que todos los condicionales se deben a los posibles valores que puede tener la variable $calif$. Por lo tanto, es posible convertir la cadena anidada Si-Sino usando la instrucción Dependiendo De.

**Solución:**

```
Proc: Calificaciones
  Caracter: calif
  Escriba "Calificación en letra: "
  Lea calif
  DD(calif)Haga
    Opción A :
      Escriba "La calificación numérica está entre 90 y 100"
    Fin
    Opción B :
      Escriba "La calificación numérica está entre 80 y 89.9"
    Fin
    Opción C :
      Escriba "La calificación numérica está entre 70 y 79.9"
    Fin
    Opción D :
      Escriba "Mala calificación"
    Fin
    Sino :
      Escriba "Por supuesto que no tuve nada que ver con mi calificación."
    Fin
  Fin_DD
Fin_proc
```

*Comentarios*: Se observa claramente cómo el largo y la complejidad del algoritmo en sí se tornan de menor tamaño, juntando todos los Si-Sino anidados en un solo condicional Dependiendo De.

---

Ejemplo 3.65 Fecha válida I

---

Escriba un algoritmo para desplegar los mensajes siguientes:

*Introduzca el número del mes:*
*Introduzca un día del mes:*

Haga que su programa acepte y almacene un número en la variable *mes* en respuesta al primer mensaje, y acepte y almacene un número en la variable *dia* en respuesta al segundo mensaje. Si el mes introducido no está entre 1 y 12 inclusive, imprima un mensaje informando al usuario que se ha introducido un mes inválido. Si el día introducido no está entre 1 y 31, imprima un mensaje informando al usuario que se ha introducido un día inválido.

```
Proc: Fechas válidas
| Entero: mes, dia
| Escriba "Introduzca el número del mes: "
| Lea mes
| Escriba "Introduzca un día del mes
| Lea dia
| Si(mes ≥ 1 y mes ≤ 12)Entonces
| | Si(dia ≥ 1 y dia ≤ 31)Entonces
| | | Escriba "La fecha es válida"
| | Sino
| | | Escriba "El día es inválido"
| | F.Si
| Sino
| | Escriba "El mes es inválido"
| F.Si
Fin_proc
```

---

Ejemplo 3.66 Fecha válida II

---

En un año que no es bisiesto (febrero tiene 28 días), enero, marzo, mayo, julio, agosto, octubre y diciembre tienen 31 días y todos los demás meses tienen 30. Usando esta información modifique el programa escrito en el ejercicio anterior para desplegar un mensaje cuando se introduce un día inválido para un mes introducido por un usuario. Para este algoritmo ignore los años bisiestos.

```
Proc: Fecha válida
| Entero: mes, dia
| Escriba "Introduzca un mes (use 1 para Ene, etc.): "
| Lea mes
| Escriba "Introduzca un día del mes
| Lea dia
| Si(mes ≥ 1 y mes ≤ 12)Entonces
| | Si(mes = 1 o 3 o 5 o 7 o 8 o 10 o 12)Entonces
| | | Si(dia ≥ 1 y dia ≤ 31)Entonces
| | | | Escriba "La fecha",dia,"/",mes,"es válida"
| | | Sino
| | | | Escriba "El día es inválido"
| | | F.Si
| | Sino
| | | Si(mes = 2)Entonces
| | | | Si(dia ≥ 1 y dia ≤ 28)Entonces
| | | | | Escriba "La fecha ", dia, "/", mes, " es válida"
1 2 3 4 5
```

1 2 3 4 5
**Sino**
**Escriba** "El día es inválido"
**F.Si**
**Sino**
**Si**($dia \geq 1$ y $dia \leq 28$)**Entonces**
**Escriba** "La fecha ", $dia$, "/", $mes$, " es válida"
**Sino**
**Escriba** "El día es inválido"
**F.Si**
**F.Si**
**F.Si**
**Sino**
**Escriba** "El mes es inválido"
**F.Si**
**Fin_proc**

---

Ejemplo 3.67 Determinar el cuadrante I

El cuadrante en el que reside una línea trazada desde el origen es determinado por el ángulo que forma la línea con el eje $x$ positivo como sigue:

| **Ángulo desde el eje $x$ positivo** | **Cuadrante** |
|---|---|
| Entre 0 y 90 grados | I |
| Entre 90 y 180 grados | II |
| Entre 180 y 270 grados | III |
| Entre 270 y 360 grados | IV |

Usando esta información escriba un algoritmo que acepte el ángulo de la línea como una entrada del usuario y determine y despliegue el cuadrante apropiado a los datos introducidos. (Nota: Si el ángulo tiene exactamente 0, 90, 180 o 270 grados, la línea no reside en ningún cuadrante sino que se encuentra en un eje).

**Proc**: Cuadrante
**Real:** $ang$
**Escriba** "Introduzca el ángulo de la línea: "
**Lea** $ang$
**Si**($ang \geq 0$ y $ang \leq 360$)**Entonces**
**Si**($ang = 90$ o $ang = 180$ o $ang = 270$ o $ang = 0$ o $ang = 360$)**Entonces**
**Escriba** "La línea está sobre uno de los ejes"
**Sino**
**Si**($ang > 0$ y $ang < 90$)**Entonces**
**Escriba** "La línea se encuentra en el I cuadrante"
1 2 3 4

```
1 2 3 4
      Sino
        Si(ang > 90 y ang < 180)Entonces
          Escriba “La línea se encuentra en el II cuadrante”
        Sino
          Si(ang > 180 y ang < 270)Entonces
            Escriba “La línea se encuentra en el III cuadrante”
          Sino
            Escriba “La línea se encuentra en el IV cuadrante”
          F.Si
        F.Si
      F.Si
    F.Si
  Sino
    Escriba “NO introdujo un ángulo válido”
  F.Si
Fin_proc
```

---

Ejemplo 3.68 Determinar el cuadrante II

---

Modifique el algoritmo anterior de modo que se despliegue un mensaje que identifique un ángulo de cero grados como el eje $x$ positivo, un ángulo de 90 grados como el eje $y$ positivo, un ángulo de 180 grados como el eje $x$ negativo $y$ un ángulo de 270 grados como el eje $y$ negativo.

*Análisis*: Entre la primera y segunda parte de este ejercicio con una simple combinación del condicional Si y un condicional Dependiendo De no se produce un mayor cambio en la estructura del algoritmo.

```
Proc: Cuadrante
  Escriba “Introduzca el ángulo de la línea: ”
  Entero: ang
  Lea ang
  Si(ang ≥ 0 y ang ≤ 360)Entonces
    Si(ang = 90 o ang = 180 o ang = 270 o ang = 0 o ang = 360)Entonces
      DD(ang)Haga
        Opción 90 :
          Escriba “La línea se encuentra sobre el eje y positivo”
        Fin
        Opción 180 :
          Escriba “La línea se encuentra sobre el eje x negativo”
        Fin
        Opción 270 :

1 2 3 4 5
```

1 2 3 4 5
**Escriba** "La línea se encuentra sobre el eje $y$ negativo"
**Fin**
**Opción** 360 :
**Escriba** "La línea se encuentra sobre el eje $x$ positivo"
**Fin**
**Opción** 0 :
**Escriba** "La línea se encuentra sobre el eje $x$ positivo"
**Fin**
**Fin_DD**
**Sino**
**Si**($ang > 0$ y $ang < 90$)**Entonces**
**Escriba** "La línea se encuentra en el I cuadrante"
**Sino**
**Si**($ang > 90$ y $ang < 180$)**Entonces**
**Escriba** "La línea se encuentra en el II cuadrante"
**Sino**
**Si**($ang > 180$ y $ang < 270$)**Entonces**
**Escriba** "La línea se encuentra en el III cuadrante"
**Sino**
**Escriba** "La línea se encuentra en el IV cuadrante"
**F.Si**
**F.Si**
**F.Si**
**F.Si**
**Sino**
**Escriba** "NO introdujo un ángulo válido"
**F.Si**
**Fin_proc**

Ejemplo 3.69 Tarifa de registro

Con base en el año del modelo y el peso de un automóvil el estado de Nueva Jersey determina la clase de vehículo y la tarifa de registro que le corresponde usando la siguiente tabla:

| **Año del modelo** | **Peso** | **Clase de peso** | **Tarifa** |
|---|---|---|---|
| 1970 o anterior | Menos de 2700 lbs | 1 | $16.50 |
| | 2700 a 3800 lbs | 2 | 25.50 |
| | Más de 3800 lbs | 3 | 46.50 |
| 1971 a 1979 | Menos de 2700 lbs | 1 | 27.00 |
| | 2700 a 3800 lbs | 2 | 30.50 |
| | Más de 3800 lbs | 3 | 52.50 |
| 1980 o posterior | Menos de 3500 lbs | 7 | 19.50 |
| | 3500 lbs o más | 8 | 52.50 |

Usando esta información escriba un algoritmo que acepte el año y el peso de un automovíl y determine y despliegue la clase y la tarifa de registro para el mismo.

**Entrada 1:** Año
**Entrada 2:** Peso
**Salida 1:** Clase
**Salida 2:** Tarifa
**Entrada 1 Ejemplo:** 2002
**Entrada 2 Ejemplo:** 3250
**Salida 1 Ejemplo:** 7
**Salida 2 Ejemplo:** 19.50

```
Proc: Tarifa de registro
| Entero: anio, peso
| Escriba "Introduzca el año del modelo del automovil: "
| Lea anio
| Escriba "Introduzca el peso del automovil: "
| Lea peso
|
| Si(anio ≤ 1970)Entonces
| | Si(peso < 2700)Entonces
| | | Escriba "Clase de peso: 1"
| | | Escriba "Tarifa de Registro: $16.50"
| | Sino
| | | Si(peso ≥ 2700 y peso ≤ 3800)Entonces
| | | | Escriba "Clase de peso: 2"
| | | | Escriba "Tarifa de Registro: $25.50"
| | | Sino
| | | | Escriba "Clase de peso: 3"
| | | | Escriba "Tarifa de Registro: $46.50"
| | | F.Si
| | F.Si
| Sino
| | Si(anio ≥ 1971 y anio ≤ 1979)Entonces
| | | Si(peso < 2700)Entonces
| | | | Escriba "Clase de peso: 4"
| | | | Escriba "Tarifa de Registro: $27.50"
| | | Sino
| | | | Si(peso ≥ 2700 y peso ≤ 3800)Entonces
| | | | | Escriba "Clase de peso: 5"
| | | | | Escriba "Tarifa de Registro: $30.50"
| | | | Sino
| | | | | Escriba "Clase de peso: 6"
1 2 3 4 5
```

1 2 3 4 5

**Escriba** "Tarifa de Registro: \$52.50"
**F.Si**
**F.Si**
**Sino**
**Si**($peso < 3500$)**Entonces**
**Escriba** "Clase de peso: 7"
**Escriba** "Tarifa de Registro: \$19.50"
**Sino**
**Escriba** "Clase de peso: 8"
**Escriba** "Tarifa de Registro: \$52.50"
**F.Si**
**F.Si**
**F.Si**
**Fin_proc**

---

Ejemplo 3.70 Juego del 21

---

En el juego del 21, el valor de las cartas del 2 al 10 es el que tienen impreso, sin importar de qué palo sean. Las cartas de personajes (jota, reina y rey) se cuentan como 10 y el as como 1 u 11, dependiendo de la suma de todas las cartas en una mano. El as se cuenta como 11 solo si el valor total resultante de todas las cartas en una mano no excede de 21, de lo contrario se cuenta como 1. Usando esta información escriba un algoritmo que acepte los valores de tres cartas como entradas (un 1 correspondiente a un as, un 2 a un dos, etc.), calcule el valor total de la mano en forma apropiada y despliegue el valor de las tres cartas con un mensaje impreso.

**Entrada 1**: Un número del 1 al 13
**Entrada 2**: Un número del 1 al 13
**Salida**: Valor de la mano
**Entrada 1 Ejemplo**: 1
**Entrada 2 Ejemplo**: 12
**Salida Ejemplo**: 15

**Proc**: Juego del 21
**Entero:** $total$, $carta1$, $carta2$, $carta3$
$total \leftarrow 0$
**Escriba** "Introduzca la primera carta: "
**Lea** $carta1$
**Si**($carta1 \geq 2$ y $carta1 \leq 10$)**Entonces**
$total \leftarrow total + carta1$
**Sino**

1 2

```
1 2
|  Si(carta1 ≥ 11 y carta1 ≤ 13)Entonces
|  |  total ← total + 10
|  Sino
|  |  total ← total + 11
|  F.Si
F.Si
Escriba "Introduzca la segunda carta: "
Lea carta2
Si(carta2 ≥ 2 y carta2 ≤ 10)Entonces
|  total ← total + carta2
Sino
|  Si(carta2 ≥ 11 y carta2 ≤ 13)Entonces
|  |  total ← total + 10
|  Sino
|  |  Si(total + 11 ≤ 21)Entonces
|  |  |  total ← total + 11
|  |  Sino
|  |  |  total ← total + 1
|  |  F.Si
|  F.Si
F.Si
Escriba "Introduzca la tercera carta: "
Lea carta3
Si(carta3 ≥ 2 y carta3 ≤ 10)Entonces
|  total ← total + carta3
Sino
|  Si(carta3 ≥ 11 y carta3 ≤ 13)Entonces
|  |  total ← total + 10
|  Sino
|  |  Si(total + 11 ≤ 21)Entonces
|  |  |  total ← total + 11
|  |  Sino
|  |  |  total ← total + 1
|  |  F.Si
|  F.Si
F.Si
Si(total > 21 y carta1 = 1)Entonces
|  total ← total − 10
F.Si
Si(total > 21 y carta2 = 1)Entonces
|  total ← total − 10
F.Si
1
```

1
**Escriba** "La suma total de las cartas es: ", *total*
**Fin_proc**

---

Ejemplo 3.71 Tipo de ángulo

Un ángulo es considerado agudo si es menor que 90 grados; obtuso si es mayor que 90 grados, y recto si es igual a 90 grados. Usando esta información escriba un algoritmo que acepte un ángulo, en grados, y despliegue el tipo de ángulo correspondiente a los grados introducidos.

*Análisis*: Se declara la variable *angulo* de tipo Real. Luego se pregunta en orden descendente por el valor del ángulo, es decir, si *angulo* es mayor que 90, es obtuso, si no entonces se pregunta por el siguiente valor, que correspondería a la igualdad a 90, y si no cumple las condiciones anteriores, el ángulo es agudo.

**Entrada:** Ángulo en grados
**Salida:** Tipo de ángulo
**Proc**: Tipo de ángulo
  **Entero:** *angulo*
  **Escriba** "Ingrese ángulo en grados: "
  **Lea** *angulo*
  **Si**($angulo > 90$)**Entonces**
    **Escriba** "El ángulo es obtuso"
  **Sino**
    **Si**($angulo = 90$)**Entonces**
      **Escriba** "El ángulo es recto"
    **Sino**
      **Escriba** "El ángulo es agudo"
    **F.Si**
  **F.Si**
**Fin_proc**

---

Ejemplo 3.72 Tipo de triángulo

Un triángulo es equilátero si todos sus lados son iguales; isósceles, si solo tiene dos lados iguales, y en caso contrario, escaleno. Escriba un algoritmo que dado las longitudes de los lados de un triángulo determine si es equilátero, isósceles o escaleno.

*Análisis*: Se pregunta si hay igualdad entre todos los lados, por tanto se usa un "y" (AND), luego se pregunta si algún par son iguales, por eso se usa "o" (OR), y en caso contrario se deduce que el triángulo es escaleno.

**Entrada:** Longitudes de los lados

**Salida:** Tipo de triángulo (equilátero, isósceles, escaleno)

**Entrada Ejemplo:** 80, 20, 95
**Salida Ejemplo:** Escaleno

**Proc**: Tipo de triángulo
  **Real:** $a$, $b$, $c$
  **Escriba** "Ingrese las longitudes de los lados del triángulo: "
  **Lea** $a, b, c$
  **Si**($a = b$ y $b = c$)**Entonces**
    **Escriba** "El triángulo es equilátero"
  **Sino**
    **Si**($a = b$ o $b = c$ o $a = c$)**Entonces**
      **Escriba** "El triángulo es isósceles"
    **Sino**
      **Escriba** "El triángulo es escaleno"
    **F.Si**
  **F.Si**
**Fin_proc**

---

Ejemplo 3.73 Tipo de año

Todos los años que se dividen exactamente entre 400 o que son divisibles exactamente entre cuatro y no son divisibles exactamente entre 100 son años bisiestos. Por ejemplo, en vista que 1600 es divisible exactamente entre 400, el año 1600 fue bisiesto. Del mismo modo, en vista que 1988 es divisible exactamente entre cuatro pero no entre 100, también fue bisiesto. Usando esta información escriba un algoritmo que acepte el año como una entrada del usuario, determine si el año es bisiesto y despliegue un mensaje apropiado que le indique al usuario si el año es bisiesto o no (Bronson & Borse, 2010).
*Análisis*: Gracias a las condiciones compuestas en este caso, nos podemos ahorrar varios niveles de anidamiento y hacer un algoritmo que parece largo pero que realmente es corto y sencillo. La variable *anio* representa el año.

**Entrada:** Año
**Salida:** Bisiesto o No es bisiesto
**Ejemplo: Entrada:** 2004
**Salida:** Es un año bisiesto

**Proc**: Tipo de año
  **Entero:** *anio*
  **Escriba** "Introduzca un año: "
  **Lea** *anio*
1

1
**Si**($anio$ **mod** $400 = 0$ o ($anio$ **mod** $4 = 0$ y $anio$ **mod** $100 \neq 0$))**Entonces**
  **Escriba** "El año ", $anio$, " es bisiesto"
**Sino**
  **Escriba** "El año ", $anio$, " NO es bisiesto"
**F.Si**
**Fin_proc**

---

Ejemplo 3.74 Número de cifras

Realice un algoritmo que diga cuántos dígitos tiene un número.

*Análisis*: Se declaran las variables enteras $n, nd$ y $cn$; donde $n$ será el número que se va a evaluar, $nd$ el número de dígitos y $cn$ la variable que se usará para realizar las divisiones enteras entre 10.

Se utiliza un condicional Si-Sino anidado en el Sino con un ciclo Mientras que. En el primer condicional se evalúa si el número es 0; si lo es, al número de dígitos se le asigna 1. En el Sino se obtiene el valor absoluto del número y se hacen divisiones sucesivas en el ciclo Mientras que siempre que el valor entero del número sea entero, en cada iteración se incrementa el número de dígitos y al finalizar el ciclo se escribe el resultado.

**Entrada:** Un número entero.
**Salida:** El número de dígitos del número proporcionado.

2345 $\Longrightarrow$ **A** $\Longrightarrow$ El número 2345 tiene 4 dígitos.

**Proc**: Número de cifras
  **Entero:** $nd$, $n$, $cn$
  $nd \leftarrow 0$
  **Escriba** "Por favor digite un número entero"
  **Lea** $n$
  **Si**($n = 0$)**Entonces**
    $nd \leftarrow 1$
  **Sino**
    $cn \leftarrow abs(n)$
    **Mq**($cn > 0$)**Haga**
      $cn \leftarrow cn\, DIV\, 10$
      $nd \leftarrow nd + 1$
    **Fin_Mq**
  **F.Si**
  **Escriba** $n$, "tiene", $nd$, "dígitos"
**Fin_proc**

Ejemplo 3.75 Test de número primo

Realice un algoritmo que lea un entero y diga si es primo o no.

*Análisis*: Se declaran las variables enteras $n, j$ y la booleana $sw$, donde $n$ será el número que se va a verificar si es primo o no y $j$ y $sw$ servirán de control para el ciclo Mientras que.

Se utiliza un ciclo Mientras que, que se ejecutará siempre que $sw$ sea falso y $j$ menor o igual a la mitad del número $n$. En el interior del ciclo se evalúa si entre los números del 2 a $n/2$ existe un divisor de $n$; si es así, se detendrá el ciclo y se escribirá que el número no es primo; de lo contrario, si no se encontró ningún divisor, entonces el ciclo se detendrá porque $j$ llegó a $n/2$, y en este caso se escribirá que el número $n$ es primo.

Por otra parte, vea que si $d$ es un divisor de $n$, entonces $\frac{n}{d}$ también lo es. A su vez, si $d$ no fuese un divisor de $n$, entonces $\frac{n}{d}$ tampoco lo sería. Esto quiere decir que podemos agrupar los divisores de $n$ en parejas, es decir,

$$\left(d_1, \frac{n}{d_1}\right), \left(d_2, \frac{n}{d_2}\right), \left(d_3, \frac{n}{d_3}\right), \ldots, \left(d_k, \frac{n}{d_k}\right).$$

Sin embargo, para evitar que se repitan podemos agregar una condición adicional, y es que cada divisor $d_k$ sea menor que su pareja $\frac{n}{d_k}$. Ahora bien, si

$$d_k \leq \frac{n}{d_k} \implies d_k^2 \leq n \implies d_k \leq \sqrt{n}.$$

De lo anterior se puede concluir que si se desea revisar todos los divisores de $n$ o revisar si $n$ es un número primo (es decir, no tiene divisores), basta con revisar hasta los números menores o iguales que $\sqrt{n}$. Si encuentra que $a$ es un divisor, entonces $\frac{n}{a}$ también lo será.

La variable $j$ que se utilizará como contador se declara en 2, puesto que para saber si un número es primo o no interesa saber si tiene un divisor distinto de 1 y de el mismo.

**Entrada:** Un número natural.
**Salida:** Diremos si el número proporcionado es primo o no.

$$9 \implies \boxed{\mathbf{A}} \implies \text{El número 9 no es primo}$$

**Proc**: Número primo
| **Entero:** $j$, $n$
| **Booleano:** $sw$
| $j \leftarrow 2$
| $sw \leftarrow falso$
| **Escriba** "Por favor digite un número entero"
| **Lea** $n$
| **Mq**($j <= (n/2)$ y $sw = falso$)**Haga**
| | // La condición $j \leq n/2$ puede ser remplazada por $j \leq \sqrt{n}$
| | **Si**($n$ **mod** $j = 0$)**Entonces**
| | | $sw \leftarrow verdadero$
| | **F.Si**
| | $j \leftarrow j + 1$
| **Fin_Mq**
| **Si**($sw = falso$)**Entonces**
| | **Escriba** $n$, "es primo"
| **Sino**
| | **Escriba** $n$, "no es primo"
| **F.Si**
**Fin_proc**

---

Ejemplo 3.76 Número perfecto

Realice un algoritmo que diga si un número es perfecto o no, es decir, que la suma de sus divisores propios es igual a el mismo número.
*Análisis*: Se declaran las variables enteras $n, j$ y $suma$; donde $n$ será el número natural de entrada, $j$ la variable utilizada para controlar el ciclo Mientras que y $suma$ la variable en la que se acumularán las sumas de los divisores.

Se realiza un ciclo Mientras que desde 2 hasta la mitad del número (ya que un número no tendrá divisores mayores a su mitad), se usa un condicional en cada iteración para evaluar si $j$ es divisor de $n$, y si es así, se acumulará en la variable suma. Al finalizar el ciclo se compara $n$ con el valor de suma, y si son iguales se escribe que el número es perfecto, de lo contrario se escribe que el número no es perfecto.

**Entrada:** Un número natural.
**Salida:** Diremos si el número proporcionado es perfecto o no.

6 $\Longrightarrow$ **A** $\Longrightarrow$ El número 6 es perfecto

**Proc**: Número perfecto
|
1

```
1
| Entero: j, suma, n
| j ← 2, suma ← 1
| Escriba "Por favor digite un número entero"
| Lea n
| Mq(j <= (n/2))Haga
| | Si(n mod j = 0)Entonces
| | | suma ← suma + j
| | F.Si
| | j ← j + 1
| Fin_Mq
| Si(suma = n)Entonces
| | Escriba n, "es perfecto"
| Sino
| | Escriba n, "no es perfecto"
| F.Si
Fin_proc
```

---

Ejemplo 3.77 Números amigos

---

Realice un algoritmo que diga si dos números son amigos o no. Dos números son amigos cuando la suma de sus divisores es igual al otro, respectivamente.

*Análisis*: Se declaran las variables enteras $n, m, sum, sum2$ y $j$; donde $n$ y $m$ son los dos números que se va a verificar si son o no amigos, $j$ servirá para controlar los dos ciclos Mientras que, $sum$ y $sum2$ serán las variables en las que se acumule la suma de los divisores de los números $m$ y $n$, respectivamente.

Se utilizan dos ciclos Mientras que: uno en el que $j$ va desde 2 hasta la mitad de $m$ y el otro de 2 hasta la mitad de $n$; en ambos ciclos se pregunta si $j$ es divisor del número; si lo es, se acumula en $sum$ o $sum2$. Al finalizar se evalúa si la suma de los divisores de $m$ ($sum$) es igual a $n$ y si la suma de los divisores de $n$ ($sum$) es igual a $m$; si es así, se escribe que los números son amigos, de lo contrario que no lo son.

**Entrada:** Dos números enteros.
**Salida:** Un mensaje que indique si los números son pares o impares.

$$6, 8 \implies \boxed{\textbf{A}} \implies \text{Los números 6 y 8 no son amigos}$$

```
Proc: Números amigos
| Entero: sum, sum2, j, m, n
| sum ← 1, sum2 ← 1, j ← 2
1
```

```
1
  Escriba "Por favor digite dos números enteros"
  Lea m, n
  Mq(j <= (m/2))Haga
    Si(m mod j = 0)Entonces
      sum ← sum + j
    F.Si
    j ← j + 1
  Fin_Mq
  j ← 2
  Mq(j <= (n/2))Haga
    Si(n mod j = 0)Entonces
      sum2 ← sum2 + j
    F.Si
    j ← j + 1
  Fin_Mq
  Si(sum = n y sum2 = m)Entonces
    Escriba m, n, "son amigos"
  Sino
    Escriba m, n, "no son amigos"
  F.Si
Fin_proc
```

Ejemplo 3.78 Números de Fibonacci

Realice un algoritmo que muestre el $n$-ésimo término de la serie de Fibonacci. La serie de Fibonnaci se obtiene mediante la siguiente función:

*Análisis*: Primero se declaran las variables $n$, $sig$, $ant$, $preant$ y $j$ de tipo Entero; la variable $n$ es el número que se pide por pantalla; la variable del control del ciclo Para es $j$; la variable $preant$ es inicializada en 0, puesto que es el primer término de la serie de Fibonacci; la variable $ant$ es inicializada en 1, como segundo término de la serie, y la variable $sig$ es la que contiene el término que se calcule.

Después se pide digitar el número por pantalla, y se captura su valor. Se verifica si $n = 0$ o $n = 1$; en este caso, el valor de la función es 0. Si $n = 2$, el valor de la función es $0 + 1$. En caso de que $n > 2$, se van calculando los términos en el ciclo Para; el término siguiente en la función es la suma del penúltimo y el último término. Luego de calcular el término se escribe en pantalla y a la variable $preant$ se le asigna el valor de la variable $ant$, y a esta se le asigna el valor de la variable $sig$, que corresponde al último término calculado.

$$F(n) = \begin{cases} 0, & \text{si } n = 0 \\ 1, & \text{si } n = 1 \\ F(n-1) + F(n-2), & \text{si } n > 1 \end{cases}$$

5 $\Longrightarrow$ **A** $\Longrightarrow$ 1 1 2 3 5

```
Proc: Números de Fibonacci
  Entero: n, preant, ant, sig
  Escriba "Digite n"
  Lea n
  preant ← 1
  ant ← 2
  Si(n = 1)Entonces
    Escriba preant
  Sino
    Si(n = 2)Entonces
      Escriba preant, ant
    Sino
      Escriba preant, ant
      Para(j = 3, n, 1)Haga
        sig ← ant + preant
        Escriba sig
        preant ← ant
        ant ← sig
      Fin_Para
    F.Si
  F.Si
Fin_proc
```

Ejemplo 3.79 Divisores de un número

Realice un algoritmo que reciba un número entero y dé como salida sus divisores.

**Entrada:** Un número natural
**Salida:** Todos los divisores del número proporcionado por el usuario.

28 $\Longrightarrow$ **A** $\Longrightarrow$ Los divisores de 28 son: 1, 2, 4, 7, 14, 28.

*Análisis*: Los divisores de un número son aquellos que al dividir el número por uno de ellos, el residuo es cero, por lo tanto se hará uso de la función **mod**. Para obtenerlos basta con llegar a su mitad, y siempre se encontrarán el 1 y él mismo. Como se necesita que la división se haga por lo menos una vez (entre 1) se emplea el ciclo Haga Hasta (HH), que se efectuará hasta que sobrepase la mitad. La variable que recorre de uno en uno los números para saber si son divisores es $j$. Si al efectuar '*num* **mod** $j$' se obtiene cero, entonces es divisor, por tanto se muestra por pantalla.

**Proc**: Divisores de un número
  **Entero:** $j$, $n$
  $j \leftarrow 1$
  **Escriba** "Por favor digite un número entero"
  **Lea** $n$
  **Escriba** "Los divisores de ", $n$, " son: "
  **HH**
    **Si**($n$ **mod** $j = 0$)**Entonces**
      **Escriba** $j$
    **F.Si**
    $j \leftarrow j + 1$
  **Fin_HH**($j > (n/2)$)
  **Escriba** $n$
**Fin_proc**

---

Ejemplo 3.80 Cubos de Nicomaco de Gerasa

Nicomaco de Gerasa descubrió la siguiente propiedad de los números naturales: Al sumar el primer impar se obtiene el primer cubo:
$1 = 1^3 = 1$.

Al sumar los dos siguientes impares se obtiene el segundo cubo:
$3 + 5 = 2^3 = 8$.

Al sumar los tres siguientes impares se obtiene el tercer cubo:
$7 + 9 + 11 = 3^3 = 27$.

Al sumar los cuatro siguientes impares se obtiene el cuarto cubo:
$13 + 15 + 17 + 19 = 4^3 = 64$.
etc...

Con esta propiedad calcule y muestre los cubos de los primeros $n$ números naturales.

*Análisis*: Las variables que se van a usar son las siguientes: $n$ es el número de cubos que se va a calcular; *base* es el valor al que en un momento determinado se le está calculando el cubo; *impar* contendrá el número impar que se va a sumar; $i$ es la variable de control del ciclo Para interior y permite saber cuántos valores se han sumado para una misma base; *resultado* guarda la suma, que al final queda con el cubo correspondiente a la base. El algoritmo debe iniciar pidiendo la cantidad de cubos que se va a calcular. Un ciclo Para externo controla el número de cubos que se debe mostrar. Un ciclo interior debe sumar tantos impares como la base indique, por lo tanto, la suma de los impares se controla con un ciclo Para que va desde 1 hasta *base* con contador $i$. En su interior se debe actualizar el valor de *impar* de 2 en 2 y, a su vez, se acumula el valor en la variable *resultado*. Vea que *resultado* debe ser iniciado en 0 cada vez que se va a calcular un nuevo cubo.

Por ejemplo, si $n = 4$, la salida debe ser como se muestra a continuación:

$$4 \Longrightarrow \boxed{\mathbf{A}} \Longrightarrow \begin{array}{c} 1^3 = 1 \\ 2^3 = 8 \\ 3^3 = 27 \\ 4^3 = 64 \end{array}$$

**Proc**: Cubos de Nicomaco de Gerasa
  **Entero:** *impar*, *base*, *n*, *i*, *resultado*
  $impar \leftarrow 1$
  **Escriba** "Digite el número de cubos que desea ver: "
  **Lea** $n$
  **Para**$(base = 1, n, 1)$**Haga**
    **Escriba** $base$, "^3 = "
    // Se inicia resultado en 0
    $resultado \leftarrow 0$
    **Para**$(i = 1, base, 1)$**Haga**
      $resultado \leftarrow resultado + impar$
      $impar \leftarrow impar + 2$
    **Fin_Para**
    **Escriba** $resultado$
  **Fin_Para**
**Fin_proc**

---

Ejemplo 3.81 Potenciación

---

Dado una base $b$ y un exponente $e$, determine el valor de $b^e$.

*Análisis*: Se usará una variable $p$ que almacenará el resultado; esta se declara en 1; de tal forma que si se hace la instrucción $p \leftarrow p * b$ repetidas veces se obtiene una potencia de $b$. Esta instrucción se debe controlar mediante un ciclo Para, de tal forma que se haga $e$ veces, y así obtener $b^e$.

**Proc**: Potenciación
  **Entero:** *p*, *b*, *e*, *i*
  $p \leftarrow 1$
  **Lea** $b, e$
  **Para**$(i = 1, e, 1)$**Haga**
    $p \leftarrow p * b$
  **Fin_Para**
  **Escriba** "El resultado es", $p$
**Fin_proc**

---

Ejemplo 3.82 Factorial de muchos números

Calcule el factorial justo después de leído una serie de números. Se detiene cuando se ingresa un número negativo.

*Análisis*: Primero ilustremos la situación con un ejemplo. Si el usuario ingresa $3, 4, 5, 1, -1$, entonces la salida debe ser $6, 24, 120, 1$. Se usa un ciclo Haga Hasta de tal manera que por lo menos se ejecute una vez, dado que al menos el usuario debe ingresar un número. Se valida si el número es un entero positivo. Luego se inicializa la variable factorial en 1, puesto que es el módulo de la multiplicación. Se calcula su factorial y se muestra. En caso de no tratarse de un entero positivo, el algoritmo termina.

```
Proc: Factorial de muchos números
| Entero: factorial, num, i
| HH
| | factorial ← 1
| | Escriba "Ingrese número"
| | Lea num
| | Si(num ≥ 0)Entonces
| | | Para(i = 1, num, 1)Haga
| | | | factorial ← factorial * i
| | | Fin_Para
| | | Escriba "Factorial: ", factorial
| | F.Si
| Fin_HH(num < 0)
Fin_proc
```

Ejemplo 3.83 Múltiplos de 3

Escriba un algoritmo que seleccione y despliegue los primeros 20 números enteros que sean divisibles entre 3.

*Análisis*: Se presentarán 3 soluciones. La primera consiste en ir buscando número por número, y en el momento en que uno de esos sea divisible por 3 incrementa un contador $i$ en 1; cuando haya encontrado 19 números, entra en el Mientras que, debido a que se cumple la condición $i = 19 < 20$. Por último, encuentra el vigésimo múltiplo de 3; allí $i = 20$ y no se cumple que sea menor que 20, por tanto no vuelve a entrar en el ciclo, y finaliza el algoritmo.

```
Proc: Múltiplos de 3
| Entero: i, num
| i ← 0
1
```

```
1
  num ← 1
  Mq(i < 20)Haga
    Si(num mod 3 = 0)Entonces
      Escriba num
      i ← i + 1
    F.Si
    num ← num + 1
  Fin_Mq
Fin_proc
```

La segunda solución usa el hecho de que un Para puede hacer saltos de 3 en 3, de tal manera que salte de múltiplo en múltiplo y simplemente los muestre en pantalla; se puede observar que el incremento en este caso es 3.

```
Proc: Múltiplos de 3
  Entero: i
  Para(i = 3, 60, 3)Haga
    Escriba i
  Fin_Para
Fin_proc
```

La tercera solución es la más versátil, pues mostrará 20 números en pantalla, que serán los primeros 20 múltiplos de 3, pues en el Escriba se encuentra la expresión $i * 3$, por tanto, el primer número que se va a mostrar será 3, el siguiente 6, y así sucesivamente hasta mostrar los primeros 20 múltiplos de 3.

```
Proc: Múltiplos de 3
  Entero: i
  Para(i = 1, 20, 1)Haga
    Escriba i * 3
  Fin_Para
Fin_proc
```

---

Ejemplo 3.84 Factores primos

Realice un algoritmo que muestre los factores primos de un número $n$.

$$78 \implies \boxed{\mathbf{A}} \implies 2, 3, 13$$

*Análisis*: El algoritmo utiliza dos variables: el número al que se le hallarán los factores primos $n$ y un índice $i$ que funcionará como divisor. Para hallar los factores primos de un número se divide el número por otro tantas veces como sea posible, es decir, mientras que el residuo sea cero, hecho que se realiza en el Mientras que interno; cuando no se pueda seguir dividiendo por

este número se avanzan dos cantidades, para repetir el proceso hasta que el número sea menor o igual a uno. Al dividir un número, por ejemplo 2, tantas veces como sea posible, este número no será divisible por 4, ni por 8, ni por 16, y así sucesivamente. Por lo tanto, si se entra en el Mientras que, el número con el que se entra es primo. Por consiguiente, en su interior se asigna a una variable booleana *sw* el valor de 1, que permitirá escribir el número si es primo y divide a $n$.

Además, primero se debe revisar el 2 y luego el resto de los números, de esta forma se evita ir de uno en uno, y se va de dos en dos, lo cual reduce a la mitad el número de cálculos que realizará el algoritmo. Lo anterior se debe a que no es necesario revisar si un número es par, puesto que ya se dividió el número por 2 tantas veces como fue posible, entonces el número no será divisible por un número par.

Por último, observe que se hace necesario el uso de estructuras Mientras que porque no se conoce cuántas veces se hará la división.

```
Proc: Factores primos
|  Entero: n, sw, i
|  Escriba "Digite un número"
|  Lea n
|  // Primero se va a dividir el número por 2 como sea posible
|  // tantas veces como sea posible
|  sw ← 0
|  Mq(n mod 2 = 0)Haga
|  |  sw ← 1
|  |  n ← n/2
|  Fin_Mq
|  Si(sw = 1)Entonces
|  |  Escriba i, ", "
|  F.Si
|  i ← 3
|  // Se procede a revisar los otros números
|  Mq(n > 1)Haga
|  |  Mq(n mod i = 0)Haga
|  |  |  sw ← 1
|  |  |  n ← n/i
|  |  Fin_Mq
|  |  i ← i + 2
|  |  Si(sw = 1)Entonces
|  |  |  Escriba i, ", "
|  |  F.Si
|  Fin_Mq
Fin_proc
```

Ejemplo 3.85 Media y varianza

Realice un algoritmo que lea un número de datos y calcule la media y la varianza de estos.

Media:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} a_i = (a_1 + ... + a_n)/n$$

Varianza:

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n} = \left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right) - \overline{x}^2$$

Por ejemplo,

$$1\ 2\ 3 \implies \boxed{\mathbf{A}} \implies \begin{array}{c}\text{Media: } 2\\ \text{Varianza: } 0.66\end{array}$$

*Análisis*: Se usarán las variables $n$ y $j$ para el número de datos que se van a computar y para controlar el ciclo Para, respectivamente. En la variable $x$ se almacenará el dato ingresado por el usuario; $med$ y $var$ son la media y la varianza, respectivamente. Se utiliza un ciclo Para que va de 1 hasta el total de datos $n$ que sirve para leer los datos e ir calculando la sumatoria de las fórmulas de la media y la varianza. Al finalizar el ciclo se divide $med$ entre $n$ y se tiene la media de los datos; también se realizan los cálculos restantes para obtener la varianza y se imprimen ambas por pantalla.

```
Proc: Media y varianza
 | Real: med, var, x
 | Entero: n, j
 | med ← 0, var ← 0
 | Escriba "Por favor, digite el número de datos"
 | Lea n
 | Para(j = 1, n, 1)Haga
 |  | Escriba "Digite el dato número", j
 |  | Lea x
 |  | med ← med + x
 |  | var ← var + x * x
 | Fin_Para
 | med ← med/n
 | var ← ((var/n) − med * med)
 | Escriba "La media de los ", n, "datos es:", med
 | Escriba "La varianza de los ", n, "datos es:", var
Fin_proc
```

Ejemplo 3.86 Serie de Taylor de $sin(x)$

Realice un algoritmo que calcule el seno de un ángulo utilizando la serie de Taylor del seno; dicha serie está definida de la siguiente forma:

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

*Análisis*: La variable $n$ determina hasta qué término se va a evaluar la serie. En realidad, no se necesitan infinitos términos para evaluar la serie. Con 10 términos que se evalúen se cuenta con una muy buena aproximación del seno de un ángulo. La variable *seno* contendrá la suma de los términos; *pi* es la constante $\pi$, la cual se declara en 3.1416. Se le pedirá al usuario que ingrese el ángulo al cual se le desea calcular su seno. El usuario ingresa el ángulo en grados, por lo tanto se hace necesaria una conversión, puesto que $x$ en la serie debe estar en radianes. Recuerde que para pasar de grados a radianes el ángulo debe multiplicarse por $pi/180$.

La variable $vf$ corresponde al exponente de $-1$ y al exponente de $x$ en la serie. A su vez, hay que calcular el factorial de $vf$. En la variable $fact$ se almacena el cálculo del factorial de $vf$ utilizando un ciclo Para que se ejecuta en cada iteración del ciclo Para externo y va desde 1 hasta $vf$; al finalizar este ciclo Para, a la variable *termino* se le asigna $(-1)^j$ para calcular el signo positivo o negativo, multiplicado por $x^{vf}/fact$; como se tiene calculado el término, se adiciona al total de *seno*. Cuando finaliza el ciclo Para externo se escribe el resultado.

$$45 \implies \boxed{\mathbf{A}} \implies \sin 45 = 0.7071$$

**Proc**: Cálculo de la serie de Taylor de la función $\sin x$

```
| Entero: n, j, fact, vf
| Real: seno, pi, x, termino
| n ← 50
| seno ← 0, pi ← 3.1416
| Escriba "Escriba el ángulo en grados: "
| Lea x
| x ← pi * x/180
| Para(j = 0, n, 1)Haga
| | fact ← 1
| | vf ← 2 * j + 1
| | Para(i = 1, vf, 1)Haga
| | | fact ← fact * i
| | Fin_Para
1 2
```

1 2
*termino* ← $(-1)^j * x^{vf}/fact$
*seno* ← *seno* + *termino*
**Fin_Para**
**Escriba** "sin(", $x$, ") = ", *seno*
**Fin_proc**

Observe que en cada iteración se están haciendo cálculos de más. Vea que se puede usar el término anterior para calcular el siguiente. Para ello note que

$$\frac{(-1)^n}{(2n+1)!}x^{2n+1} = \frac{-1}{(2n+1)(2n)}x^2 \cdot \frac{(-1)^{n-1}}{(2n-1)!}x^{2n-1}$$

Por lo tanto, no es necesario estar calculando el factorial de $vf$ en cada iteración. El primer término de la serie es

$$\frac{(-1)^0}{(2 \cdot 0+1)!}x^{2\cdot 0+1} = x. \tag{3.1}$$

A continuación se muestra una nueva solución usando lo anterior.

**Proc**: Cálculo de la serie de Taylor de la función $\sin x$
**Entero:** $n$, $j$
**Real:** *seno*, *pi*, $x$, *termino*
$n \leftarrow 50$
*seno* ← 0, *pi* ← 3.1416
**Escriba** "Escriba el ángulo en grados: "
**Lea** $x$
$x \leftarrow pi * x/180$
*termino* ← $x$
*seno* ← *termino*
**Para**($j = 1, n, 1$)**Haga**
// Se actualiza la variable *termino* según la expresión (3.1)
*termino* ← *termino* $* (-1) * x^2/((2*j+1)*(2*j))$
*seno* ← *seno* + *termino*
**Fin_Para**
**Escriba** "sin(", $x$, ") = ", *seno*
**Fin_proc**

---

Ejemplo 3.87 Serie de Taylor de $\exp(x)$

Realice un algoritmo que calcule la serie exponencial; dicha serie está definida por la serie de Taylor de $\exp x$:

$$e^x = \sum_{j=0}^{\infty} \frac{x^j}{j!}$$

*Análisis*: De forma similar al problema anterior, primero se va a identificar cómo calcular el término $n$-ésimo en función del anterior, es decir, del $n-1$. Observe que

$$\frac{x^n}{n!} = \frac{x}{n} \cdot \frac{x^{n-1}}{(n-1)!} \tag{3.2}$$

La variable *termino* contiene el término actual de la serie. Observe que el primer término de la serie es $x^0/0! = 1$.

Por ejemplo, si $x$ es igual a uno, el algoritmo debería mostrar el valor del número de Euler, $e$, o aproximadamente 2.7182.

$$1 \implies \boxed{\textbf{A}} \implies e^1 = 2.7182818$$

**Proc**: Serie de Taylor de la función exponencial
  **Entero:** $n$, $j$
  **Real:** *exp*, $x$, *termino*
  $n \leftarrow 50$
  $exp \leftarrow 0$
  **Escriba** "Escriba el valor de $x$: "
  **Lea** $x$
  $termino \leftarrow 1$
  $exp \leftarrow 1$
  **Para**$(j = 1, n, 1)$**Haga**
    // Se actualiza la variable *termino* según la expresión (3.1)
    $termino \leftarrow termino * x/j$
    $exp \leftarrow exp + termino$
  **Fin_Para**
  **Escriba** "exp(", $x$, ") = ", *exp*
**Fin_proc**

Observe que la estructura del algoritmo es la misma que la del cálculo de la serie de Taylor de la función sin($x$).

---

Ejemplo 3.88 Pi

---

El valor del número $\pi$ se puede calcular con la precisión deseada sabiendo que la serie infinita del matemático alemán Leibnitz se define como

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} \ldots \text{ tiende a } \frac{\pi}{4}$$

Escriba un algoritmo que calcule una aproximación al número $\pi$ usando un número de términos suministrado por el usuario.

*Análisis*: La variable $n$ será el número de términos que se generarán de la serie; $j$ será la variable que servirá para controlar el ciclo Para que va desde 1 hasta $n$; *termino* es la variable en la que se almacena el valor de la serie que se calcula en cada iteración; la variable *signo* será la que indique si el término es positivo o negativo y se va alternando cada vez; *impar* es la variable en la que se almacenan los impares para ir generando la serie; *pi* será la suma de todos los términos.

Se utiliza un ciclo Para en el que en cada iteración se calcula un término de la serie y se van sumando hasta tener $n$ términos, y es entonces cuando finaliza el ciclo; después se multiplica $pi * 4$, ya que la serie genera a $pi/4$, y por último se escribe el resultado de la aproximación.

*Ejemplo*: $N = 5 \Longrightarrow \frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} = 3.33968$

$$5 \Longrightarrow \boxed{\textbf{A}} \Longrightarrow 3.33968$$

Observe que este problema es similar a los problemas anteriores pero más sencillo, puesto que es más fácil calcular el término nuevo en función del anterior.

**Proc**: Pi
  **Entero:** *signo*, *impar*, *n*
  **Real:** *pi*, *termino*
  $signo \leftarrow -1, impar \leftarrow 1$
  $pi \leftarrow 0, termino \leftarrow 0$
  **Escriba** "Escriba el número de términos para la aproximación: "
  **Lea** $n$
  **Para**$(j = 1, n, 1)$**Haga**
    $signo \leftarrow signo * -1$
    $termino \leftarrow signo * (1/impar)$
    $pi \leftarrow pi + termino$
    $impar \leftarrow impar + 2$
  **Fin_Para**
  $pi \leftarrow pi * 4$
  **Escriba** "La aproximación de pi es: ", *pi*
**Fin_proc**

---

Ejemplo 3.89 Juego de cartas

---

Realice un algoritmo que simule un juego de cartas entre dos jugadores; gana el que muestre el valor mayor 3 veces consecutivas.

*Análisis*:

Jugador 1: 3 5 7
Jugador 2: 2 4 6 $\Longrightarrow \boxed{\textbf{A}} \Longrightarrow$ El ganador es el jugador 1

Primero se utilizarán las variables $i$ y $j$ y se declararan en 0. En ellas se almacenará el número de veces que gana cada jugador, respectivamente. Se utilizará un ciclo Mientras que, puesto que no se conoce el número de iteraciones totales que se va a ejecutar por el ciclo. En él se leerán las dos cartas, se pregunta cuál de las dos es mayor y se aumenta el contador del jugador respectivo; posteriormente se reinicia el contador del jugador contrario; el ciclo se acaba cuando uno de los dos contadores llegue a 3. De esta forma, solo se puede ganar cuando se consiga tres victorias consecutivas.

```
Proc: Juego de cartas
| Entero: j, i, a, b
| j ← 0, i ← 0
| Mq(i ≠ 3 y j ≠ 3)Haga
| | Escriba "Digite el valor de carta jugador 1:"
| | Lea a
| | Escriba "Digite el valor de carta jugador 2:"
| | Lea b
| | Si(a > b)Entonces
| | | i ← i + 1
| | | j ← 0
| | F.Si
| | Si(b > a)Entonces
| | | j ← j + 1
| | | i ← 0
| | F.Si
| Fin_Mq
| Si(i = 3)Entonces
| | Escriba "¡EL JUGADOR 1 ES EL GANADOR!"
| Sino
| | Si(j = 3)Entonces
| | | Escriba "¡EL JUGADOR 2 ES EL GANADOR!"
| | F.Si
| F.Si
Fin_proc
```

---

Ejemplo 3.90 Función real

Haga un algoritmo que reciba un número real y su salida sea

$$f(x) = \begin{cases} \frac{\pi}{2} + 7, & \text{si } x \geq 20 \\ x^2, & \text{si } 0 \leq x < 20 \\ x^8 - x^5, & \text{si } x < 0 \end{cases}$$

*Análisis*: Para hallar la salida del algoritmo simplemente se debe utilizar

el condicional Si y para cada condición se escribe la salida correcta. Por ejemplo:

$$\begin{array}{rcccl} 21 & \Longrightarrow & & \Longrightarrow & \frac{\pi}{2}+7 \\ 15 & \Longrightarrow & \boxed{\mathbf{A}} & \Longrightarrow & 15^2 \\ -21 & \Longrightarrow & & \Longrightarrow & -21^8 - -21^5 \end{array}$$

**Proc**: Función real
 **Real:** $x$
 **Escriba** "Por favor, digite un número real"
 **Lea** $x$
 **Si**($x >= 0$)**Entonces**
  **Si**($x >= 20$)**Entonces**
   **Escriba** "f(x)= pi/2+7*", $x$
  **Sino**
   **Escriba** "f(x)= ", $x$, "ˆ2"
  **F.Si**
 **Sino**
  **Escriba** "f(x)= ", $x$, "ˆ8 − ", $x$, "ˆ 5"
 **F.Si**
**Fin_proc**

---

Ejemplo 3.91 Ver si un dígito está o no

---

Realice un algoritmo que lea un número y diga si contiene un dígito $d$.

*Análisis*: En el algoritmo se emplean las siguientes variables: $n$ para guardar el número que se va a verificar; $cn$ es una copia de $n$, donde está el dígito que se va a buscar; $d$ es el dígito a buscar; $dig$ guarda cada dígito a contrastar, y una variable booleana; $sw$, inicializada en falso y que solo cambia su valor en caso que el número sí contenga al dígito.

Por ejemplo,

$$\begin{array}{lcccl} n = 5489, \quad d = 9 & \Longrightarrow & \boxed{\mathbf{A}} & \Longrightarrow & \text{el dígito 9 está en el número.} \\ n = 5484, \quad d = 9 & \Longrightarrow & & \Longrightarrow & \text{el dígito 9 NO está en el número.} \end{array}$$

Para saber si $d$ está contenido se debe ir dividiendo el número entre 10 y sacando el residuo módulo 10 (si se divide entre 10, el residuo será el último dígito). Luego se compara este residuo con el dígito que se está buscando. Dicha división sucesiva se realiza por medio de un Mientras que hasta que se encuentre el dígito, colocando la variable $sw$ en 1, o hasta que el número sea cero. En este último caso quiere decir que el dígito $d$ no está contenido, por tanto, el $sw$ se mantiene en falso. Al final se imprime el resultado.

```
Proc: Ver si un dígito esta o no
    Entero: sw, n, d, cn, dig
    sw ← 0
    Escriba "Por favor, digite el número y el dígito que se va a buscar"
    Lea n, d
    cn ← n
    Mq(n > 0 y sw = 0)Haga
        dig ← n mod 10
        Si(dig = d)Entonces
            sw ← 1
        F.Si
        n ← n/10
    Fin_Mq
    Si(sw = 1)Entonces
        Escriba "El número", cn, "contiene el dígito ", d
    Sino
        Escriba "El número", cn, "no contiene el dígito ", d
    F.Si
Fin_proc
```

---

Ejemplo 3.92 El dígito 2

Haga un algoritmo que lea un grupo de números de cuatro dígitos, cuente los que contienen el número dos y sume los que no lo contienen.

*Análisis*: Suponga que el usuario ingresa los números 2358, 2587, 3548 y 3597.

$2358, 2587, 3548, 3597 \implies \boxed{\mathbf{A}} \implies$ Números que contienen al dos: 2 / Suma de los que no lo tienen: 7145.

La salida es esa, puesto que hay dos números que contienen el dos (2358 y 2587), mientras que los números que no lo contienen (3548 y 3597) suman 7145.

Para el algoritmo se utilizarán las siguientes variables: $n$ para la cantidad de datos que se van a leer, $num$ para cada número leído; $d1$, $d2$, $d3$, $d4$ para guardar los dígitos que componen el número leído y dos variables acumulativas; $cont$ que llevará el registro de cuántos contienen al dígito 2, y $sum$ la suma de los que no lo contienen.

Se utiliza un ciclo Para que va desde uno hasta la cantidad de datos que se leerán. Por cada número leído se evalúa si contiene o no al 2; para esto se separan sus dígitos; como el enunciado dice que solo contiene 4, entonces en $d1$ se guarda la unidad; para esto se aplica la operación **mod** del número

con 10, en $d2$ la decena; para esto es necesario primero dividir el número por 10 para luego obtener el residuo módulo 10; de igual forma se obtiene la centena, asignada a $d3$, pero esta vez dividiéndolo por 100; por último, para sacar la unidad de mil, que se guarda en $d4$, basta con dividir el número entre 1000. Cuando se tiene todos los dígitos se procede a verificarlos, y si por lo menos uno es igual a 2, entonces se incrementa *cont* en 1; si no a la suma que se lleva se le agrega el número.

```
Proc: Dígito 2
  Entero: cont, sum, j, d1, d2, d3, d4, n, num
  cont ← 0, sum ← 0
  Escriba "Digite el número de datos"
  Lea n
  Para(j = 1, n, 1)Haga
    Escriba "Digite el dato número", j
    Lea num
    d1 ← num mod 10
    d2 ← (num DIV 10) mod 10
    d3 ← (num DIV 100) mod 10
    d4 ← num DIV 1000
    Si(d1 = 2 o d2 = 2 o d3 = 2 o d4 = 2)Entonces
      cont ← cont + 1
    Sino
      sum ← sum + num
    F.Si
  Fin_Para
  Escriba "Los números que contienen el dígito dos son: ",cont
  Escriba "La suma de los números que no contienen el dígito dos es: ", sum
Fin_proc
```

---

Ejemplo 3.93 Calculadora

Cree un algoritmo que permita sumar, multiplicar y dividir dos números pero que no permita la división por 0 y despliegue un mensaje apropiado cuando se intente dicha división.

*Análisis*: Se utilizarán las variables $num1$ y $num2$ para almacenar los números digitados por el usuario. La variable *cod* almacenará la opción digitada por el usuario: 1 para sumar, 2 para multiplicar y 3 para dividir. Una estructura Dependiendo De escribirá los resultados correspondientes a la operación seleccionada por el usuario. En el caso de la división, antes de realizarla se verifica si $num2 \neq 0$. En caso de que el usuario no ingrese una opción entre las mostradas, se despliega un mensaje.

```
Proc: Calculadora que no permite la división por 0
  Real: num1, num2
  Entero: cod
  Escriba "Digite el primer número":
  Lea num1
  Escriba "Digite el segundo número":
  Lea num2
  Escriba "1 para Sumar"
  Escriba "2 para Multiplicar"
  Escriba "3 para Dividir"
  Escriba "Seleccione la operación a realizar: "
  Lea cod
  DD(cod)Haga
    Opción '1' :
      Escriba num1, "+", num2, "=", num1 + num2
    Fin
    Opción '2' :
      Escriba num1, "*", num2, "=", num1 * num2
    Fin
    Opción '3' :
      Si(num2 ≠ 0)Entonces
        Escriba num1, "/", num2, "=", num1/num2
      Sino
        Escriba "La división por 0 no está definida. Inténtelo nuevamente."
      F.Si
    Fin
    Sino :
      Escriba "No seleccionó una opción válida."
    Fin
  Fin_DD
Fin_proc
```

---

Ejemplo 3.94 Ecuación cuadrática

---

Realice un algoritmo que dado los tres coeficientes (a, b, c) muestre el resultado de la ecuación cuadrática de la forma $ax^2 + bx + c$. Asuma que $|a| + |b| > 0$.

*Análisis*: Las variables $a$, $b$ y $c$ son los coeficientes de una ecuación cuadrática; las variables $x1$ y $x2$ son las raíces de la misma y dependerán del discriminante de la ecuación, el cual se almacena en la variable $dis$. Con el discriminante se determina si $x1$ y $x2$ son reales o complejas.

Se le solicita al usuario que ingrese en las variables $a, b, c$ los coeficientes de la ecuación. Se calcula el discriminante y se verifica si es negativo o positivo,

y las raíces serán complejas o reales, respectivamente. Finalmente se escribe el valor de las raíces.

Por ejemplo, si $a = 3, b = 2, c = 1$, entonces el algoritmo debe mostrar $x_1 = -\frac{1}{3} - \frac{\sqrt{2i}}{3}$ y $x_2 = -\frac{1}{3} + \frac{\sqrt{2i}}{3}$.

**Proc**: Ecuación cuadrática
  **Real:** $a$, $b$, $c$, $dis$
  **Escriba** "Por favor, digite los coeficientes de la ecuación: $a$, $b$ y $c$"
  **Lea** $a, b, c$
  $dis \leftarrow b^2 - 4 * a * c$
  **Si**($dis < 0$)**Entonces**
    **Escriba** "$x1 =$", $-b/2 * a$, " + ", $\sqrt{-dis/2a}$, "i"
    **Escriba** "$x2 =$", $-b/2 * a$, " - ", $\sqrt{-dis/2a}$, "i"
  **Sino**
    **Escriba** "$x1 =$", $-b + \sqrt{dis/2a}$
    **Escriba** "$x2 =$", $-b - \sqrt{dis/2a}$
  **F.Si**
**Fin_proc**

---

Ejemplo 3.95 Suma de Gauss

---

Realice un algoritmo que lea un número entero y muestre la suma desde el 1 hasta el número digitado; demuestre que es igual al resultado de $\frac{n(n+1)}{2}$ (Fórmula de Gauss)

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + ... + n = \frac{n\,(n+1)}{2}$$

*Análisis*: La variable $n$ se utilizará para almacenar el número que se pide por pantalla; la variable del control del ciclo Para es $j$; a la variable *suma* se le asigna el cálculo de la sumatoria con la fórmula, y la variable $s$ contiene la suma calculada sumando los términos desde 1 hasta $n$. Las variables *suma* y $s$ son inicializadas en 0. Por último, se escribe el valor de $s$ y *suma*.

3 $\Longrightarrow$ **A** $\Longrightarrow$ La suma es igual a 6

**Proc**: Verificación de la Fórmula de Gauss
  **Entero:** $s$, $n$
  $s \leftarrow 1$
  **Escriba** "Por favor, digite un número entero"
  **Lea** $n$
1

```
1
  Si(n = 1)Entonces
    Escriba 1
  Sino
    Escriba 1
    Para(j = 2, n, 1)Haga
      s ← s + j
      Escriba s
    Fin_Para
  F.Si
Fin_proc
```

Ejemplo 3.96 Newton-Raphson

Haga un algoritmo que calcule la raíz cuadrada de un número suministrado por el usuario, utilizando el algoritmo de Newton. $A$ es el número que digitó el usuario (es decir, al que se le desea hallar la raíz) y $X$ representa el valor que se le va a mostrar al usuario (es decir, la raíz cuadrada de $A$), que inicialmente puede ser igual a $A$. Realice este proceso hasta que $|X_{n+1} - X_n|$ sea menor que una tolerancia $tol$ ingresada por el usuario.

$$X_{n+1} = X_n + \frac{1}{2}\left(\frac{A}{X_n} - X_n\right)$$

*Análisis*: Observe que en este algoritmo se calcula el valor de $X_{n+1}$ en función del valor anterior, es decir, $X_n$, por lo tanto se requiere una “semilla” o un valor de inicio. Se tomará como $X_0 = A/2$ una primera aproximación al valor de la raíz de $A$. Ahora bien, este proceso se debe repetir hasta que $|X_{n+1} - X_n| < tol$. La variable $xn$ contendrá la iteración anterior, mientras que la variable $xn1$ contendrá la iteración actual. La variable $dif$ contiene la diferencia entre dos aproximaciones consecutivas.

Por ejemplo, si el usuario ingresa 9, el algoritmo debe mostrar 3.

$$9 \implies \boxed{\mathbf{A}} \implies 3$$

El algoritmo mostrado corresponde a un caso particular del algoritmo de Newton-Raphson. En general, si se tiene una ecuación de la forma $f(x) = 0$, una manera de aproximar la solución de esta en función de una aproximación anterior $x_n$ es

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

**Proc**: Cálculo de la raíz de un número
  **Real:** $a$, $tol$, $xn$, $xn1$, $dif$
  **Escriba** "Digite un número real: "
  **Lea** $a$
  **Escriba** "Digite la tolerancia: "
  **Lea** $tol$
  $xn \leftarrow a$
  $xn1 \leftarrow xn + 0.5 * (a/xn - xn)$
  $dif \leftarrow |xn - xn1|$
  **Mq**($dif \geq tol$)**Haga**
    $xn \leftarrow xn1$
    $xn1 \leftarrow xn1 + 0.5 * (a/xn1 - xn1)$
    $dif \leftarrow |xn - xn1|$
  **Fin_Mq**
  **Escriba** "La raíz cuadrada de ", $a$, " es igual a ", $xn$
**Fin_proc**

---

Ejemplo 3.97 Calificaciones

---

Cuatro enteros entre 0 y 100 representan las puntuaciones de un estudiante de un curso de Algoritmia. Escriba un algoritmo para encontrar la media de estas puntuaciones y visualice una tabla de notas de acuerdo con el siguiente cuadro:

| **Media** | **Puntuación** |
|---|---|
| $90 - 100$ | $A$ |
| $80 - 89$ | $B$ |
| $70 - 79$ | $C$ |
| $60 - 69$ | $D$ |
| $0 - 59$ | $E$ |

*Análisis*: Una vez leída una calificación, se debe determinar qué puntuación le corresponde. Como se ha visto anteriormente, la mejor forma de hacerlo es preguntar en orden. Por lo tanto, se pregunta si la calificación es mayor que 89; en caso contrario, si es mayor que 79, y así sucesivamente. Se repite el proceso para las otras tres calificaciones con el uso de un ciclo Para, y en cada Si-Sino se almacena la letra que corresponde en una variable de tipo carácter; al final se escribe la calificación. Lo anterior evita colocar un Escriba en el interior de cada Si-Sino. Por último, se acumula en *prom* la suma de los enteros correspondientes a cada calificación, para luego ser dividido entre 4. Este último resultado corresponde a la media aritmética.

```
Proc: Calificaciones
  Real: prom
  Entero: i, n, entero
  Caracter: cal
  prom ← 0
  Lea n
  Para(i = 1, n, 1)Haga
    Escriba "Por favor, ingrese la nota ", i, ": "
    Lea entero
    Si(entero > 89)Entonces
      cal ← A
    Sino
      Si(entero > 79)Entonces
        cal ← B
      Sino
        Si(entero > 69)Entonces
          cal ← C
        Sino
          Si(entero > 59)Entonces
            cal ← D
          Sino
            cal ← E
          F.Si
        F.Si
      F.Si
    F.Si
    Escriba "Calificación ", i,": ", cal
    prom ← prom + entero
  Fin_Para
  prom ← prom/n
  Escriba "La media de las calificaciones es: ", prom
Fin_proc
```

---

Ejemplo 3.98 Fórmula de Herón

El área de un triángulo cuyos lados son $a, b, c$ se puede calcular con la fórmula de Herón:

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

donde $s$ es el semiperímetro del triángulo y está dado por $s = (a+b+c)/2$. Escriba un algoritmo que lea las longitudes de los tres lados de un triángulo y calcule el área del triángulo.

*Análisis*: Observe que no se le pide al algoritmo verificar que las longitudes de los lados sean números reales positivos; por lo tanto, el algoritmo corresponde a solamente lectura de datos y asignación.

Por otro lado, si $a, b, c$ son los lados del triángulo, según el teorema del coseno:

$$\cos C = \frac{a^2 + b^2 - c^2}{2ab}$$

donde $C$ es el ángulo formado por los lados $a$ y $b$. Sin embargo, la bien conocida identidad $\sin^2 C + \cos^2 C = 1$ implica

$$\sin C = \sqrt{1 - \cos^2 C} = \sqrt{\frac{4a^2b^2 - (a^2 + b^2 - c^2)^2}{2ab}}.$$

Por último, recuerde que el área de este triángulo está dada por $A = \frac{1}{2}ab\sin C$. Remplazando y reorganizando términos se llega a la fórmula de Herón.

Por ejemplo, si la longitud de los lados es 3, 4 y 5, entonces el semiperímetro es $s = \dfrac{3+4+5}{2} = 6$. Luego, $s(s-a)(s-b)(s-c) = 6 \cdot (6-3)(6-4)(6-5) = 36$. Finalmente, el área del triángulo es $A = \sqrt{36} = 6$.

**Proc**: Fórmula de Herón
 **Real:** $a$, $b$, $c$, $p$, $area$
 **Escriba** "Ingrese los lados del triángulo: "
 **Lea** $a, b, c$
 $p \leftarrow (a + b + c)/2$
 $area \leftarrow \sqrt{p(p-a)(p-b)(p-c)}$
 **Escriba** "El área del triángulo es ", $area$
**Fin_proc**

---

Ejemplo 3.99 Formato de fecha

Escriba un programa que acepte tres números enteros, los cuales corresponderán a: el primero, el día; el segundo, el mes; el tercero, el año de una fecha. Y a continuación muestre en formato '*dia* de *mes* de *año*' para visualizar el mes.

*Análisis*: Dado que no se está pidiendo verificar si la fecha es válida, se asumirá que el usuario ingresa una fecha válida. A su vez, puesto que el mes se ingresa como un número entero, entonces hay que determinar a qué mes corresponde. Por otro lado, resulta más conveniente usar un Dependiendo De que anidar cláusulas Si-Sino. Por ejemplo, si el usuario ingresa 1, 1, 1999, la salida debe ser "1 de enero de 1999".

**Proc**: Formato de fecha
 **Entero:** $a$, $b$, $c$
 **Escriba** "Ingrese los números correspondientes a la fecha: "
 **Lea** $a, b, c$
1

```
1
DD(b)Haga
    Opción 1 :
        Escriba a, " de enero de ", c
    Fin
    Opción 2 :
        Escriba a, " de febrero de ", c
    Fin
    Opción 3 :
        Escriba a, " de marzo de ", c
    Fin
    Opción 4 :
        Escriba a, " de abril de ", c
    Fin
    Opción 5 :
        Escriba a, " de mayo de ", c
    Fin
    Opción 6 :
        Escriba a, " de junio de ", c
    Fin
    Opción 7 :
        Escriba a, " de julio de ", c
    Fin
    Opción 8 :
        Escriba a, " de agosto de ", c
    Fin
    Opción 9 :
        Escriba a, " de septiembre de ", c
    Fin
    Opción 10 :
        Escriba a, " de octubre de ", c
    Fin
    Opción 11 :
        Escriba a, " de noviembre de ", c
    Fin
    Opción 12 :
        Escriba a, " de diciembre de ", c
    Fin
Fin_DD
Fin_proc
```

---

Ejemplo 3.100 Redondeo

Un archivo de datos contiene los cuatro dígitos, $A, B, C, D$, de un entero positivo $N$. Se desea redondear $N$ a la centena más próxima y visualizar la

salida. Por ejemplo, si $A$ es 2, $B$ es 3, $C$ es 6 y $D$ es 2, entonces $N$ será 2362 y el resultado redondeado será 2400. Si $N$ es 2300, el resultado será 2300. Diseñe el algoritmo correspondiente.

*Análisis*: Una vez leídos $A, B, C, D$, se tiene el número $N = \overline{ABCD}$, y su representación decimal corresponde a $1000A + 100B + 10C + D$. Se quiere aproximar a su centena más cercana, entonces $C$ y $D$ serían 0 y a $B$ se le suma 1 o se deja igual según sea el caso. Pero si $B$ es 9, esta suma no se ve afectada, pues $B$ se vuelve 0 y $A$ se incrementa en 1. Entonces $N1$ y $N2$ son las aproximaciones por encima y por debajo del número $N$, respectivamente; dependiendo de las restas $N1 - N$ y $N - N2$ se decide qué centena es más próxima a $N = \overline{ABCD}$.

**Proc**: Redondeo
  **Entero:** $A$, $B$, $C$, $D$, $N$, $N1$, $N2$
  **Escriba** "Ingrese los dígitos A,B,C,D: "
  **Lea** $A, B, C, D$
  $N \leftarrow 1000 * A + 100 * B + 10 * C + D$
  $N1 \leftarrow 1000 * A + (B + 1) * 100$
  $N2 \leftarrow 1000 * A + B * 100$
  **Si**$(N1 - N \leq N - N2)$**Entonces**
    **Escriba** $N1$
  **Sino**
    **Escriba** $N2$
  **F.Si**
**Fin_proc**

---

Ejemplo 3.101 Ecuaciones lineales

Un sistema de ecuaciones lineales

$$\begin{aligned} ax + by &= c \\ dx + ey &= f \end{aligned}$$

se puede resolver con las siguientes fórmulas:

$$\begin{aligned} x &= \frac{ce - bf}{ae - bd} \\ y &= \frac{af - cd}{ae - bd} \end{aligned}$$

Diseñe un programa que lea los coeficientes $a, b, c, d$ y $f$ (en ese orden) y muestre los valores de $x$ e $y$.

*Análisis*: Se deben las variables y se hacen las asignaciones correspondientes de acuerdo con las fórmulas presentadas.

**Proc**: Ecuaciones lineales
**Real:** $a, b, c, d, e, f, x, y$
**Escriba** "Ingrese los valores de a,b,c,d,e,f: "
**Lea** $a, b, c, d, e, f$
$x \leftarrow (c * e - b * f)/(a * e - b * d)$
$y \leftarrow (a * f - c * d)/(a * e - b * d)$
**Escriba** "El valor de $x$ es: ", $x$
**Escriba** "El valor de $y$ es: ", $y$
**Fin_proc**

---

Ejemplo 3.102 Fabricante

Cada unidad de disco en un embarque de estos dispositivos tiene estampado un código del 1 al 4, el cual indica el fabricante de la unidad como sigue:

| **Código** | **Fabricante de la unidad de disco** |
|---|---|
| 1 | 3M Corporation |
| 2 | Maxell Corporation |
| 3 | Sony Corporation |
| 4 | Verbatim Corporation |

Escriba un algoritmo que acepte el número de código como una entrada y con base en el valor introducido despliegue el fabricante de la unidad de disco correcto.

*Análisis*: La mejor forma de resolver este problema es utilizando la primitiva Dependiendo De: simplemente se le pide al usuario que digite el código del fabricante y luego se establecen las opciones del caso; de esta forma, dependiendo del código del fabricante, este escribirá la empresa asociada al código.

**Proc**: Fabricantes
**Entero:** *cod*
**Escriba** "Código del Fabricante: "
**Lea** *cod*
**DD**(*cod*)**Haga**
**Opción** 1 :
**Escriba** "La unidad de disco fue fabricada por 3M Corporation"
**Fin**
**Opción** 2 :
**Escriba** "La unidad de disco fue fabricada por Maxell Corporation"
**Fin**
**Opción** 3 :
**Escriba** "La unidad de disco fue fabricada por Sony Corporation"
**Fin**
1 2

1 2

**Opción** 4 :

**Escriba** “La unidad de disco fue fabricada por Verbatim Corporation”

**Fin**

**Sino** :

**Escriba** “No digitó un código válido”

**Fin**

**Fin_DD**

**Fin_proc**

---

Ejemplo 3.103 Serie armónica

---

Calcule la suma de la serie armónica

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{N}$$

donde $N$ es un número que se introduce por teclado.

*Análisis*: Se utilizarán una variable $N$ para almacenar la cantidad de términos por sumar; la variable $i$ controlará el ciclo y la variable *suma* almacenará el resultado de la serie. Es importante resaltar que la variable *suma* es inicializada en cero porque corresponde a una variable que cumple la función de acumular.

**Proc**: Serie armónica

**Real:** *suma*

**Entero:** $n$, $i$

*suma* $\leftarrow 0$

**Escriba** “Ingrese el numero N: ”

**Lea** $n$

**Para**$(i = 1, n, 1)$**Haga**

*suma* $\leftarrow$ *suma* $+ 1/i$

**Fin_Para**

**Escriba** “Resultado: ”, *suma*

**Fin_proc**

---

Ejemplo 3.104 Promedio de datos I

---

Se llevaron a cabo cuatro experimentos, cada uno consistente en seis resultados de prueba. Escriba un algoritmo que lea los seis datos por experimento y muestre el promedio de los datos por cada experimento.

*Análisis*: Se utilizarán las variables $dato, prom, suma$ para los datos, el promedio y la suma de los datos, respectivamente. A su vez, dos ciclos anidados, uno para recorrer los experimentos y otro para leer los datos de cada uno.

El ciclo externo controla el número de experimentos y el ciclo interno controla la lectura de los datos de cada experimento. Los datos leídos de un experimento se utilizan para calcular el promedio, dividiendo el acumulador *suma* entre el número de datos leídos, que es 6, y se escribe este promedio. A la variable *suma* se le asigna cero antes de iniciar la lectura de datos de cada experimento, para no acumular valores de los experimentos anteriores que alteren el resultado. La variable *i* presente en cada Escriba permite enumerar los experimentos de forma que se diferencien cuando se escriben las salidas.

```
Proc: Promedio de datos I
  Entero: i, j
  Real: suma, dato, prom
  Para(i = 1, 4, 1)Haga
    suma ← 0
    Para(j = 1, 6, 1)Haga
      Escriba "Escriba dato ", j, " del experimento ", i
      Lea dato
      suma ← suma + dato
    Fin_Para
    prom ← suma/6
    Escriba "Promedio experimento ", i, ": ", prom
  Fin_Para
Fin_proc
```

---

Ejemplo 3.105 Promedio de datos II

---

Modifique el algoritmo anterior de modo que pueda introducirse un número diferente de resultados de prueba para cada experimento. El usuario primero debe introducir el número de pruebas de dicho experimento.

*Análisis*: Para este ejercicio, el número de iteraciones en el ciclo interior está controlado por información ingresada por el usuario. Por lo tanto, se deben agregar las instrucciones para leer el número de datos de cada experimento; esto se hará en la variable *num*; y en el ciclo interior, el promedio es calculado dividiendo la variable *suma* entre la variable *num*, que indica el número de datos leídos por experimento.

```
Proc: Promedio de datos II
  Entero: i, j, num
  Real: suma, dato, prom
  Para(i = 1, 4, 1)Haga
    Escriba "Ingrese numero de datos del experimento ", i
    Lea num
    suma ← 0
1 2
```

```
1 2
│ │ Para(j = 1, num, 1)Haga
│ │ │ Escriba "Escriba dato del experimento ", i
│ │ │ Lea dato
│ │ │ suma ← suma + dato
│ │ Fin_Para
│ │ prom ← suma/num
│ │ Escriba "Promedio experimento ", i, ": ", prom
│ Fin_Para
Fin_proc
```

---

Ejemplo 3.106 Menor número triangular mayor que $n$

---

Encuentre el número natural $N$ más pequeño tal que la suma de los $N$ primeros números exceda una cantidad introducida por el teclado.

*Análisis*: En este ejercicio, la variable $N$ determinará el límite de la suma acumulada en la variable *suma*. Se utiliza un ciclo Mientras que, y se van sumando todos los números naturales hasta que el valor de la variable *suma* supere el valor de la variable $N$. En este momento se conocería el valor de $N$. Por ejemplo, si la cantidad introducida es 25, entonces la salida debe ser $N = 7$, puesto que $1+2+3+4+5+6+7 = 28 > 25$ y $1+2+3+4+5+6 = 21 < 25$.

```
Proc: Menor número triangular
│ Entero: i, suma, n
│ i ← 1
│ suma ← 1
│ Escriba "Ingrese el número N:"
│ Lea n
│ Mq(suma ≤ n)Haga
│ │ i ← i + 1
│ │ suma ← suma + i
│ Fin_Mq
│ Escriba i
Fin_proc
```

---

Ejemplo 3.107 Número de Euler

---

El valor del número de Euler, $e$, puede aproximarse usando la fórmula

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \cdots$$

Esta serie se conoce como la expansión de Taylor de la función $y = e^x$ alrededor de $x = 0$. Usando esta fórmula escriba un algoritmo que aproxime

el valor de $e$ usando un ciclo Para con 50 iteraciones, es decir, se evaluará la serie hasta los primeros 50 términos.

*Análisis*: Se declara la variable real *euler* en 0, que funcionará como un acumulador de los términos que tienen la forma $k = \frac{1}{k!}$. En el ciclo interno se calcula el factorial y se guarda en la variable $fact$. A la variable $fact$ se le asigna 1 antes de calcular el factorial del término correspondiente, para no acumular resultados de las iteraciones anteriores. Recuerde que cuando una variable acumula el resultado de la suma de varios términos debe asignársele el módulo de la suma: 0. De manera similar, cuando una variable acumula el resultado de una multiplicación de varios términos, se le asigna el módulo de la multiplicación: 1. Por otro lado, en el ciclo interno, la variable de control $j$ debe ir hasta el valor de $i$, puesto que indica la cantidad de términos que deben multiplicarse para obtener el factorial correspondiente. Luego de calcular el factorial se obtiene el término que corresponde a la iteración $i$, diviendo 1 entre el valor del factorial y acumulando el resultado en la variable *euler*.

```
Proc: Número de Euler
  Real: euler, termino
  Entero: i, j, fact
  euler ← 0
  termino ← 0
  Para(i = 0, 49, 1)Haga
    fact ← 1
    Para(j = 1, i, 1)Haga
      fact ← fact * j
    Fin_Para
    termino ← 1/fact
    euler ← euler + termino
  Fin_Para
Fin_proc
```

Como solución alterna, se puede evitar el anidamiento, teniendo en cuenta que el factorial de un número $k$ es: $k! = (k-1)! \cdot k$, siendo $k \in \mathbb{N}$. Se puede observar que la variable $fact$ en cada iteración del ciclo Para incrementa de tal forma que para la iteración $k$-ésima, $fact$ es igual a $k!$.

```
Proc: Número de Euler
  Real: euler, termino
  Entero: i, fact
  euler ← 1
  termino ← 0
  fact ← 1
  Para(i = 1, 49, 1)Haga
    fact ← fact * i
    termino ← 1/fact
1 2
```

1 2
$euler \leftarrow euler + termino$
**Fin_Para**
**Fin_proc**

---

Ejemplo 3.108 Fahrenheit a Celsius

---

Escriba un algoritmo que despliegue una tabla de 20 conversiones de temperatura de Fahrenheit a Celsius. La tabla deberá comenzar con un valor Fahrenheit de 20 grados e incrementarse en valores de 4 grados. Recuerde que

$$°\mathrm{C} = \frac{5}{9}(°\mathrm{F} - 32)$$

*Análisis*: Se utilizan las variables $Celc$ y $Fahr$ para almacenar los valores de salida por cada iteración. En el ciclo Para, la variable $Fahr$ es inicializada en 20, y el incremento es de 4, de acuerdo con el enunciado del algoritmo. El límite del ciclo es noventa y seis, porque al ir incrementando de 4 en 4 los 20 valores de conversiones de temperatura llegan a su límite en este punto.

**Proc**: Fahrenheit a Celsius
  **Entero:** $Fahr$
  **Real:** $Celc$
  **Para**($Fahr$ =20,96,4)**Haga**
    $Celc \leftarrow (5.0/9.0) * (Fahr - 32)$
    **Escriba** "Grados Fahrenheit: ",$Fahr$," Grados Celsius: ",$Celc$
  **Fin_Para**
**Fin_proc**

En caso de que no se desee calcular el límite del Ciclo Para, se puede realizar un ciclo que inicie en 1, llegue hasta 20 y con incremento de 1. Dentro del ciclo se puede calcular a $Fahr$ como $Fahr = 4(i - 1) + 20$.

**Proc**: Fahrenheit a Celsius
  **Entero:** $i$
  **Real:** $Fahr$, $Celc$
  **Para**($i$ =1,20,1)**Haga**
    $Fahr \leftarrow 4 * (i - 1) + 20$
    $Celc \leftarrow (5.0/9.0) * (Fahr - 32)$
    **Escriba** "Grados Fahrenheit: ",$Fahr$," Grados Celsius: ",$Celc$
  **Fin_Para**
**Fin_proc**

---

Ejemplo 3.109 Dilatación térmica

La expansión de un puente de acero conforme se calienta hasta una temperatura Celsius final, $T_f$, desde una temperatura Celsius inicial, $T_0$, puede aproximarse usando la fórmula

$$\text{Aumento de longitud} = a * L * (T_f - T_0)$$

donde $a$ es el coeficiente de expansión (el cual para el acero es $11.7 \times 10^{-6}$) y $L$ es el largo del puente a la temperatura $T_0$ . Usando esta fórmula escriba un algoritmo que despliegue una tabla de longitudes de expansión para un puente de acero que tiene 7365 metros de largo a 0 grados Celsius conforme la temperatura incrementa a 40 grados en incrementos de 5 grados.

*Análisis*: Se usará un ciclo Para desde 0 hasta 40 con incremento 5. Se calcula el aumento para cada iteración y se escribe el resultado. Por otro lado, aunque la fórmula original dice que hay que restar $T_f$ y $T_0$, dado que la $T_0$ en este problema es igual a 0, siempre la resta va a hacer el valor que tenga la temperatura, $T_f$, por esto solo se usa una variable denominada $T$.

```
Proc: Dilatación térmica
  Real: a, aumento
  Entero: L, T
  a ← 11.7 * 10^-6
  L ← 7365
  Para(T = 0, 40, 5)Haga
    aumento ← a * L * T
    Escriba "Temperatura(Celsius): ", T
    Escriba "Aumento de longitud: ", aumento
  Fin_Para
Fin_proc
```

Ejemplo 3.110 Caída libre

Una pelota de golf es soltada desde un avión. La distancia $d$ que cae la pelota en $t$ segundos está dada por la ecuación $d = (\frac{1}{2})gt^2$, donde $g$ es la aceleración debida a la gravedad y es igual a 32 pies/s$^2$. Usando esta información escriba un algoritmo que despliegue la distancia que cae en cada intervalo de un segundo para diez segundos y la distancia en que cae la pelota de golf al final de cada intervalo. La salida deberá completar la siguiente tabla:

| **Tiempo** | **Distancia en el intervalo actual** | **Distancia total** |
|---|---|---|
| 0 | 0.0 | 0.0 |
| 1 | 16.0 | 16.0 |
| ⋮ | ⋮ | ⋮ |
| 10 | 304 | 1600 |

*Análisis*: Se utiliza un ciclo Para para calcular la distancia de la pelota de golf. La variable de control $t$ representa la variación del tiempo y es utilizada para calcular la salida del algoritmo. Esta variable $t$ se actualiza en cada iteración.

**Proc**: Caída libre
  **Real:** $dt$, $d$
  **Entero:** $g$, $t$
  $dt \leftarrow 0$
  $g \leftarrow 32$
  **Para**$(t = 0, 10, 1)$**Haga**
    $d \leftarrow (1/2) * g * t^2$
    $dt \leftarrow dt + d$
    **Escriba** "Tiempo: ", $t$
    **Escriba** "Distancia en el intervalo: ", $d$
    **Escriba** "Distancia Total: ", $dt$
  **Fin_Para**
**Fin_proc**

---

Ejemplo 3.111 Números de tres cifras especiales

Calcule todos los números de tres cifras tales que la suma de los cubos de las cifras sea igual al valor del número. Es decir, si $\overline{abc} = n$, entonces el algoritmo debe contar este número siempre y cuando $a^3 + b^3 + c^3 = n$.

*Análisis*: Observe que los números de tres cifras van desde 100 hasta 999, por lo tanto se usará un ciclo Para que recorrerá todos estos números. Para calcular la suma $a^3 + b^3 + c^3$ se hace necesario extraer las tres cifras del número. La cifra de las unidades se puede extraer haciendo $n \,\mathbf{mod}\, 10$, puesto que un número $n = \overline{abc}$ se puede expresar como $10\overline{ab}+c$, luego $n \,\mathbf{mod}\, 10 = c$. Recuerde que $p \,\mathbf{mod}\, q$ es el residuo de la división de $p$ entre $q$.

Para extraer la segunda cifra se puede hacer uso de la operación /, la cual es una división entera. Si se realiza $n/10$, se obtiene $\overline{ab}$. Luego se realiza **mod** 10, y se extrae $b$, es decir, $b = (n/10) \,\mathbf{mod}\, 10$. Finalmente, vea que $c = n/100$.

**Salida:** Números enteros de tres cifras.

**Proc**: Números de tres cifras especiales
  **Entero:** $n$, $a$, $b$, $c$, *cont*
  **Para**$(n = 100, 999, 1)$**Haga**
    $a \leftarrow n/100$
    $b \leftarrow (n/10)$ **mod** 10
    $c \leftarrow n$ **mod** 10
    $cont \leftarrow 0$
    **Si**$(n = a^3 + b^3 + c^3)$**Entonces**
      $cont \leftarrow cont + 1$
    **F.Si**
  **Fin_Para**
  **Escriba** "Total de números: ", *cont*
**Fin_proc**

---

Ejemplo 3.112 Probabilidad exponencial

La probabilidad $p$ de que una llamada telefónica individual dure menos de $t$ minutos puede aproximarse por la función de probabilidad exponencial

$$p = 1 - e^{-t/a}$$

donde $a$ es la duración de la llamada promedio y $e$ es el número de Euler (2.71828). Usando esta ecuación de probabilidad escriba un algoritmo que calcule y despliegue una lista de probabilidades de la duración de una llamada que dure entre 1 y 10 minutos, en incrementos de un minuto.

*Análisis*: Por ejemplo, suponiendo que la duración promedio de la llamada es 2 minutos, la probabilidad que una llamada dure menos de 1 minuto se calcula como $1 - e^{-1/2} = 0.3297$. El cálculo de las probabilidades especificadas en el algoritmo se realizan utilizando una variable $t$ que represente el tiempo y que además controle el ciclo Para. El cálculo de las probabilidades especificadas en el algoritmo se realizan utilizando una variable $t$ que represente el tiempo y que además controle el ciclo Para.

**Proc**: Probabilidad exponencial
  **Real:** $e$, $a$
  **Entero:** $t$
  $e \leftarrow 2.71828$
  **Escriba** "Introduzca la duración de una llamada promedio: "
  **Lea** $a$
  **Para**$(t = 1, 10, 1)$**Haga**
    **Escriba** "La probabilidad de que una llamada dure menos de ", $t$,
    **Escriba** "minutos es de ", $1 - e^{-t/a}$)
  **Fin_Para**
**Fin_proc**

Ejemplo 3.113 Probabilidad de Poisson

El índice de llegadas de clientes a un banco concurrido en Nueva York puede estimarse usando la función de probabilidad de Poisson

$$P(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Donde $x$ es el número de clientes que llegan por minuto; $\lambda$ es el número promedio de llegadas por minuto y $e$ el número de Euler (2.71828). Usando la función de probabilidad de Poisson escriba un algoritmo que calcule y despliegue la probabilidad de llegada de 1 a 10 clientes en cualquier minuto teniendo en cuenta que el índice de llegadas promedio es de tres clientes por minuto.

*Análisis*: Por ejemplo, si el número promedio de clientes que entran en el banco es de tres clientes por minuto, entonces $\lambda$ es igual a tres. Por tanto, la probabilidad de que un cliente llegue en cualquier minuto es

$$P(x = 1) = \frac{3^1 e^{-3}}{1!} = 0.149561$$

y la probabilidad que lleguen dos clientes en cualquier minuto es

$$P(x = 2) = \frac{3^2 e^{-3}}{2!} = 0.224454$$

En este algoritmo se utilizan un ciclo Para que controlará el número de clientes que entra al banco. La variable $fact$ se incrementa en cada ciclo; observe que no es necesario un ciclo interior para calcular el factorial. Por último, a la variable *prob* se le asigna el resultado de la fórmula expuesta en el algoritmo y se escribe la salida utilizando los valores calculados.

**Proc**: Probabilidad de Poisson
  **Real:** $e$, *prob*
  **Entero:** $lam$, $fact$, $x$
  $e \leftarrow 2.71828$
  $lam \leftarrow 3$
  $fact \leftarrow 1$
  **Para**$(x = 1, 10, 1)$**Haga**
    $fact \leftarrow fact * i$
    $prob \leftarrow (lam^x e^{-(lam)})/fact$
    **Escriba** "La probabilidad de que lleguen ", $x$,
    **Escriba** " clientes en cualquier minuto es ", *prob*
  **Fin_Para**
**Fin_proc**

Ejemplo 3.114 Formato de hora

Escriba un algoritmo que lea la hora de un día en notación de 24 horas y muestre la hora en notación de 12 horas. El programa pedirá al usuario que introduzca exactamente 4 números enteros entre 0 y 9. Así por ejemplo, las nueve en punto se introduce como 0900.

*Análisis*: Se deben leer los números correspondientes a la hora. En el condicional se hace una conversión teniendo en cuenta que un número expresado en notación decimal como $\overline{ab}$ es igual $10a+b$. Se pregunta si es mayor que 12 o no, lo cual decidirá si es a.m. o p.m. Finalmente, se escribe el resultado en formato de 12 horas. Por ejemplo, si la entrada es 1354, la salida será 1:54 p.m., y esto es porque $13 > 12$.

**Proc**: Formato de hora
  **Entero:** $a$, $b$, $c$, $d$
  **Escriba** "Ingrese los números correspondientes a la hora: "
  **Lea** $a, b, c, d$
  **Si**$(10 * a + b > 12)$**Entonces**
    **Escriba** $a * 10 + b - 12$, ":", $c$, $d$, " p.m."
  **Sino**
    **Escriba** $a * 10 + b$, ":", $c$, $d$, " a.m."
  **F.Si**
**Fin_proc**

Ejemplo 3.115 Tabla de valores para función real

Realice un algoritmo que muestre una tabla de valores dados: los intervalos, el incremento y las funciones.

*Análisis*: Para cada uno de los algoritmos se establece una estructura similar, compuesta por un ciclo Para y en la que la variable de control de cada uno es utilizada para calcular el valor de las funciones en los intervalos dados.

1. $y = 3x^5 - 2x^3 + x$ para $5 \leq x \leq 10$ en incrementos de 0.2
   **Proc**
     **Entero:** $x$
     **Para**$(x = 5, 10, 0.2)$**Haga**
       **Escriba** $x$, $3x^5 - 2x^3 + x$
     **Fin_Para**
   **Fin_proc**
2. $y = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}$ para $1 \leq x \leq 3$ en incrementos de 0.1

**Proc**
**Entero:** $x$
**Para**$(x = 1, 3, 0.1)$**Haga**
| **Escriba** $x$, $1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}$
**Fin_Para**
**Fin_proc**

3. $y = 2e^{0.8t}$ para $4 \leq t \leq 10$ en incrementos de 0.2

**Proc**
**Entero:** $x$
**Real:** $e$
$e \leftarrow 2.7183$
**Para**$(t = 4, 10, 0.2)$**Haga**
| **Escriba** $t$, $2 * e^{0.8*t}$
**Fin_Para**
**Fin_proc**

Note que la evaluación de $e^{0.8*t}$ puede realizarse con la serie de Taylor antes descrita.

---

Ejemplo 3.116 Números romanos

---

Escriba un algoritmo que acepte un año menor que 4000 escrito en cifras arábigas y visualice el año escrito en números romanos. *Nota:* Recuerde que V=5, X=10, L=50, C=100, D=500, M=1000.

*Análisis*: Antes de analizar este problema observe los siguientes ejemplos de conversión:

| | | |
|---|---|---|
| IV=4 | XL=40 | CM=900 |
| MCM=1900 | MCML=1950 | MCMLX=1960 |
| MCMXL=1940 | MCMLXXXIX=1989 | |

Un proceso de conversión exitoso sustraerá en orden descendiente las cantidades 1000, 900, 500, 400, .... El lector se preguntará: ¿por qué revisar 900? Se revisa 900 porque en los números romanos esto no se designa como $500+400$ sino como 900, es decir, no como $DCCCC$ sino como $CD$, donde $CD$ es una unidad. Lo anterior quiere decir que los números romanos tienen símbolos tanto para $100, 500$ y $10000$ como para 400 y 900. Por lo tanto, una conversión exitosa requiere que se vea a 400 y a 900 como símbolos también o como “bases” de este sistema numérico.

El algoritmo debe, por tanto, sustraer en orden decreciente. Si el número ingresado es mayor que 1000, sustrae mil tantas veces como sea necesario y se agrega ‘M’ a la salida tantas veces como se restó 1000. El número resultante es un número menor que 1000. Por lo explicado anteriormente, 900 también hace parte del sistema numérico; entonces hay que revisar si el número menor que 1000 es mayor que 900; de serlo, se sustrae 900. De manera similar se revisa para 500, 400, 100, . . ., 1.

```
Proc: Números romanos
  Entero: numero, r, i
  Escriba "Ingrese el año a convertir: "
  Lea numero
  Si(numero ≥ 1000)Entonces
    r ← n/1000
    Para(i = 1, r, 1)Haga
      Escriba "M"
    Fin_Para
    numero ← numero mod 1000
  F.Si
  Si(numero ≥ 900)Entonces
    Escriba "CM"
    numero ← numero − 900
  F.Si
  Si(numero ≥ 500)Entonces
    Escriba "D"
    numero ← numero − 500
  F.Si
  Si(numero ≥ 400)Entonces
    Escriba "CD"
    numero ← numero − 400
  F.Si
  Si(numero ≥ 100)Entonces
    r ← n/100
    Para(i = 1, r, 1)Haga
      Escriba "C"
    Fin_Para
    numero ← numero mod 100
  F.Si
  Si(numero ≥ 90)Entonces
    Escriba "XC"
    numero ← numero − 90
  F.Si
  Si(numero ≥ 50)Entonces
    Escriba "L"
    numero ← numero − 50
  F.Si
  Si(numero ≥ 40)Entonces
    Escriba "XL"
    numero ← numero − 40
  F.Si
1
```

1
**Si**($numero \geq 10$)**Entonces**
  $r \leftarrow n/10$
  **Para**($i = 1, r, 1$)**Haga**
    **Escriba** "X"
  **Fin_Para**
  $numero \leftarrow numero \, \mathbf{mod} \, 10$
**F.Si**
**Si**($numero \geq 9$)**Entonces**
  **Escriba** "IX"
  $numero \leftarrow numero - 9$
**F.Si**
**Si**($numero \geq 5$)**Entonces**
  **Escriba** "V"
  $numero \leftarrow numero - 9$
**F.Si**
**Si**($numero = 4$)**Entonces**
  **Escriba** "IV"
  $numero \leftarrow numero - 4$
**F.Si**
**Si**($numero \geq 1$)**Entonces**
  $r \leftarrow n$
  **Para**($i = 1, r, 1$)**Haga**
    **Escriba** "I"
  **Fin_Para**
  $numero \leftarrow numero - r$
**F.Si**
**Fin_proc**

---

Ejemplo 3.117 Secuencia de caracteres

Diseñe un algoritmo que produzca la siguiente salida:

```
ZYXWVTSRQPONMLKJIHGFEDCBA
YXWVTSRQPONMLKJIHGFEDCBA
XWVTSRQPONMLKJIHGFEDCBA
WVTSRQPONMLKJIHGFEDCBA
VTSRQPONMLKJIHGFEDCBA
TSRQPONMLKJIHGFEDCBA
⋮
CBA
BA
A
```

Recuerde que el valor ASCII del carácter 'Z' es 90.

*Análisis*: Una variable de tipo carácter se puede declarar de la forma $caracter \leftarrow numero$, siendo *numero* el número que corresponde al valor ASCII del carácter. Usando esto, se usarán dos ciclos Para: uno externo que controlara cada línea y uno interno que controlará la impresión de cada carácter. El ciclo Para interno inicia en $90 - i + 1$, pues para cuando $i = 1$ (la primera línea) se quiere que inicie en 90, ya que corresponde con el valor ASCII de la letra Z. Para la segunda línea, el ciclo interno empezará en 89, es decir, $Y$, y así sucesivamente. Hay 26 letras en el abecedario, entonces el carácter ASCII que corresponde a la letra A sería 65. Por consiguiente, el Para interno cuando $i = 1$ debe ir desde $j = 90$ hasta $j = 65$, que es lo que se desea; luego, para cuando $i = 2$, el Para interno debe ir desde $j = 89$ hasta $j = 65$, de tal manera que se va "eliminando" la primera letra de la linea anterior. El ciclo interior siempre va hasta 65. Finalmente se produce la salida deseada.

```
Proc
  Entero: i, j
  Caracter: a
  // Se realiza la declaración de la variable a y se hace
  // enfásis en que se trata de un carácter
  Para(i = 1, 26, 1)Haga
    Para(j = 90 − i + 1, 65, −1)Haga
      a ← j
      Escriba a
    Fin_Para
    Escriba "Salto de línea"
  Fin_Para
Fin_proc
```

---

Ejemplo 3.118 Valor máximo de un conjunto de datos

Realice un algoritmo que lea un entero positivo $n$ y a continuación lea $n$ números y despliegue tanto el valor máximo como la posición en el conjunto de números introducido donde ocurre el máximo. A su vez, también realice lo mismo pero para el mínimo valor.

*Análisis*: Se debe leer el primer número, y este se asignará como el máximo ($max$) y el mínimo ($min$). A su vez, *posmin* y *posmax* se asignan como 1. Luego se deben leer $n - 1$ números. Cada vez que se lea un número se debe verificar si es mayor que el mayor número encontrado hasta el momento. Si se da lo anterior, se actualiza el valor de *max* y el valor de *posmax*. De forma similar se hace para el mínimo.

**Proc**: Valor máximo de un conjunto de datos
**Entero:** $n$, $i$
**Real:** $num$, $max$, $posmax$, $min$, $posmin$
**Escriba** "Ingrese el número de números"
**Lea** $n$
**Escriba** "Ingrese el número 1: "
**Lea** $num$
$max \leftarrow num$
$posmax \leftarrow 1$
$min \leftarrow num$
$posmin \leftarrow 1$
**Para**$(i = 1, num - 1, 1)$**Haga**
  **Escriba** "Ingrese el numero", $i + 1$, ":"
  **Lea** $num$
  **Si**$(num > max)$**Entonces**
    $max \leftarrow num$
    $posmax \leftarrow i$
  **F.Si**
  **Si**$(num < min)$**Entonces**
    $min \leftarrow num$
    $posmin \leftarrow i$
  **F.Si**
**Fin_Para**
**Escriba** "Máximo: ", $max$
**Escriba** "Mínimo: ", $min$
**Escriba** "Posición Máximo: ", $posmax$
**Escriba** "Posición Mínimo: ", $posmin$
**Fin_proc**

---

Ejemplo 3.119 Regalos

Los padres de Yanina le prometieron darle 10 pesos cuando cumpliera 12 años de edad y duplicar esta cantidad en cada cumpleaños subsiguiente hasta que excediera los mil pesos. Escriba un algoritmo para determinar qué edad tendrá la niña cuando se le dé la última cantidad y cuánto fue la cantidad total recibida.

*Análisis*: Sea $a$ la cantidad inicial. Se sabe que $a = 10$. Para cada año se sabe que la cantidad se duplica. Si $n$ es la edad de Yanina, entonces la cantidad $2^{n-12}a$ determinará la cantidad que se le da en el cumpleaños número $n$. Por ejemplo, si $n = 12$, a Yanina le dan $2^{12-12}a = 2^0a = a$ pesos en su cumpleaños número 12. Si $n = 13$, se le da $2^{13-12}a = 2^1a = 2a$, el doble del cumpleaños anterior. El problema pide resolver el menor valor de $n$ para el cual

$$a + 2a + 2^2a + \cdots + 2^{n-12}a > 1000.$$

Además, se tiene que determinar el valor del lado izquierdo de esta desigualdad.

Se proponen dos soluciones para la resolución de este problema. En la primera se hará uso de un ciclo Mientras que; este "simulará" el paso de los años. En su interior, la cantidad regalada a Yanina se irá duplicando. La condición que controla el Mientras que permite que no entre cuando el regalo exceda los 1000 pesos. Para duplicar la cantidad de regalo se duplica la variable *regalo*, a su vez se incrementa la edad de Yanina (*edad*) en 1 y se va sumando la cantidad recibida en cada regalo en un acumulador llamado *suma*. La variable *edad* funciona como un contador. Cuando el algoritmo culmine el ciclo Mientras que, el valor de *edad* correspondería al valor de $n$ buscado.

La solución 2 muestra un ciclo Para más general, que va desde $regalo = 10$ hasta $regalo \leq 1000$. Cuando $regalo > 1000$ no se cumplirá esta condición, por tanto no entrará dentro del ciclo; el incremento en este caso es ir duplicando la variable, es decir, $regalo = 2 * regalo$. Dentro del Para se incrementa la edad en 1, y se va acumulando la cantidad total recibida en *suma*, de forma similar a la solución 1.

**Solución 1:**

```
Proc
  Entero: suma, regalo, edad
  suma ← 10
  regalo ← 10
  edad ← 12
  Mq(regalo ≤ 1000)Haga
    edad ← edad + 1
    regalo ← 2 * regalo
    suma ← suma + regalo
  Fin_Mq
  Escriba "Edad ultimo regalo: ", edad
  Escriba "Cantidad ultimo regalo: ", regalo
  Escriba "Cantidad total regalada: ", suma
Fin_proc
```

**Solución 2:**

```
Proc
  Entero: e, r, s
  // e = edad, r = regalo, s = suma
  e ← 12
  Para(r = 10, r ≤ 1000, r = r * 2)Haga
    e ← e + 1
    s ← s + r
  Fin_Para
1
```

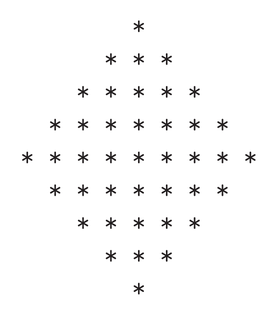
1
**Escriba** “Edad ultimo regalo: ”, $e$
**Escriba** “Cantidad ultimo regalo: ”, $r$
**Escriba** “Cantidad total regalada: ”, $s$
**Fin_proc**

---

Ejemplo 3.120 Patrón Rombo

---

Escriba un algoritmo que visualice el siguiente dibujo:



*Análisis*: En primera medida, resulta más fácil trabajar con $(n+1)/2$ una vez ingresado el valor de $n$ impar. Por lo tanto, se debe cambiar $n$ por $(n+1)/2$. Por otro lado, observe que primero se deben escribir unos espacios antes de poner un “*”, luego esos espacios se van reduciendo y vuelven aumentar a medida que escribimos más líneas. Observe que la función valor absoluto tiene estas características para valores negativos de $x$; a medida que se aumenta $x$ se disminuye su valor absoluto. Sin embargo, cuando $x > 0$, a medida que se aumenta $x$ se aumenta su valor absoluto.

Antes de continuar se ilustrará la técnica que se va a usar. Por ejemplo, suponga inicialmente que el dibujo tiene en total 5 lineas, entonces, primero se deben escribir 2 espacios y 1 asterisco; luego, 1 espacio y 3 asteriscos; luego, 0 espacios y 5 asteriscos; luego, 1 espacio y 3 asteriscos; finalmente, 2 espacios y 1 asterisco. De lo anterior, un ciclo Para externo debe controlar el salto de linea. Además, debe haber un primer ciclo Para interno, para este ejemplo, que haga primero 2 iteraciones para la primera iteración del ciclo Para externo, 1 iteración para la segunda iteración del ciclo externo, 0 iteraciones para la tercera, 1 para la cuarta y, finalmente, 2 para la quinta. Dado que $n = 5$, entonces $n = (n + 1)/2 = 3$. Entonces, si para el ciclo externo $i = 1$, entonces el contador final del primer ciclo Para debe ser 2, para $i = 2$ el contador debe ser 1, para $i = 3$ debe ser 0, para $i = 4$ debe ser 1 y para $i = 5$ debe ser 2. Pero si a $i$ se le resta $n$, es decir, 3, queda -2, -1, 0, 1, 2 para cada $i$ que va desde 1 hasta 5, pero se quieren los valores 2, 1, 0, 1, 2, entonces se aplica valor absoluto, y resulta que el número de iteraciones del primer ciclo Para interno es $|i - n|$ o $|n - i|$.

En el caso de los asteriscos, se quiere que se siga la secuencia 1, 3, 5, 3, 1,

para $i = 1, 2, 3, 4, 5$. Sin embargo, la secuencia 1, 3, 5, 3, 1, es análoga a 1, 2, 3, 2, 1, pues basta con multiplicar estos últimos por 2 y restarle 1. Ahora véase que se tiene 2, 1, 0, 1, 2. Si a 3 se le resta cada uno de estos números, se tiene $3-2, 3-1, 3-0, 3-1, 3-2$, lo cual resulta en la secuencia que se desea: 1, 2, 3, 2, 1. Finalmente, en el segundo Para se obtiene que el número de ciclos es $2*(n-|i-n|)-1$. Recuérdese que el valor de $n$ inicialmente introducido se cambia por $(n+1)/2$.

```
Proc: Patron Rombo
| Entero: n, i, j
| Escriba "Ingrese numero de filas (cantidad impar): "
| Lea n
| n ← (n + 1)/2
| Para(i = 1, 2 * n − 1, 1)Haga
| | Para(j = 1, |i − n|, 1)Haga
| | | Escriba " "
| | Fin_Para
| | Para(j = 1, 2 * (n − |i − n|) − 1, 1)Haga
| | | Escriba "*"
| | Fin_Para
| Fin_Para
Fin_proc
```

---

Ejemplo 3.121 Suma de números enteros

---

Escriba un programa que calcule la suma de los números enteros comprendidos entre 1 y 50.

*Análisis*: Se debe usar un ciclo Para con contador $i$ y un acumulador *suma* declarado inicialmente en 0, el cual acumulará la suma de los primeros 50 enteros positivos.

```
Proc: Suma de los 50 primeros enteros positivos
| Entero: suma, i
| suma ← 0
| Para(i = 1, 50, 1)Haga
| | suma ← suma + i
| Fin_Para
| Escriba "Suma: ", suma
Fin_proc
```

Escriba un algoritmo que visualice las primeras 20 potencias de 2. Suponga que no se dispone de la función potencia, es decir, no se pueden realizar operaciones de la forma $a^b$, con $a, b$ números reales.

*Análisis*: Se desea escribir las primeras 20 potencias de 2, por lo tanto se usa un ciclo Para controlado por un contador $i$, que va desde 1 hasta

20. Internamente se debe tener una variable que se vaya duplicando en cada iteración. De esta forma se crea una nueva potencia de 2 usando que $2^n = 2 \cdot 2^{n-1}$. Dado que $2^0 = 1$, se declarará una variable *pot* (que será la potencia siguiente a mostrar) en 1, y dentro del ciclo Para se debe hacer $pot = 2 * pot$. Observe que primero se debe escribir y luego duplicar, debido a que la primera potencia de 2 es 1 ($2^0$). Por último, como el ciclo hará 20 iteraciones, por tanto escribirá las primeras 20 potencias de 2.

```
Proc: 20 primeras potencias de 2
| Entero: pot, i
| pot ← 1
| Para(i = 1, 20, 1)Haga
| | Escriba pot
| | pot ← pot * 2
| Fin_Para
Fin_proc
```

Ejemplo 3.122 Incremento en el salario

En una empresa de computadoras, los salarios de los empleados se van a aumentar según su contrato actual:

| Contrato | Aumento % |
|---|---|
| 0 a 9 000 pesos | 20 |
| 9 0001 a 15 000 pesos | 10 |
| 15 001 a 20 000 pesos | 5 |
| Más de 20 000 pesos | 0 |

Escriba un algoritmo que solicite el salario actual del empleado y calcule y visualice el nuevo salario. Se introducen $N$ empleados.

*Análisis*: Dado que no introduce un número $N$ de empleados, entonces se hará uso de un ciclo Para que va desde 1 hasta $N$. En su interior se debe preguntar el contrato del empleado. Igual a como se ha hecho anteriormente, se pregunta en cierto orden. Ese orden puede ser de menor a mayor, es decir, si el contrato es menor a 9000, si no, si es menor a 15 000, y así sucesivamente. Otra forma es de mayor a menor, es decir, si el contrato es mayor a 20 000, si no, si es mayor a 15 000, y así sucesivamente. Ahora bien, dado que para mayor de 20 000 no hay aumento, entonces se puede usar una condición compuesta para preguntar si el contrato está entre 15 000 y 20 000 pesos. Por último, una vez se logra entrar a un Si, se asigna el nuevo valor de contrato de acuerdo con la tabla y se muestra dicho valor. Observe que el comando Escriba no es necesario colocarlo dentro de cada anidamiento.

**Proc**: Cálculo de nuevos salarios
|
1

```
1
  Entero: N, i, contrato
  Escriba "Ingrese numero de empleados: "
  Lea N
  Para(i = 1, N, 1)Haga
    Escriba "Ingrese contrato: "
    Lea contrato
    Si(contrato > 15000 y contrato < 20000)Entonces
      contrato ← contrato * 1.05
    Sino
      Si(contrato > 9000)Entonces
        contrato ← contrato * 1.1
      Sino
        contrato ← contrato * 1.2
      F.Si
    F.Si
    Escriba "Nuevo salario: ", contrato
  Fin_Para
Fin_proc
```

Ejemplo 3.123 Estadística de notas

Haga un algoritmo que reciba las notas de $n$ estudiantes y su salida sea la tabla de distribución estadística para las notas 1, 2, 3, 4 y 5:

1,2,4,3,5,3
1,2,3,3,4,5 ⟹ **A** ⟹
2 Estudiantes sacaron 1
2 Estudiantes sacaron 2
4 Estudiantes sacaron 3
2 Estudiantes sacaron 4
2 Estudiantes sacaron 5

*Análisis*: La variable $n$ determinará el número total de estudiantes al que se les va a leer la nota; las variables $c1$, $c2$, $c3$, $c4$, $c5$ determinarán las calificaciones; *nota* guardará temporalmente la nota ingresada por el usuario, y por último se utilizará el apuntador $j$ para el ciclo. Una vez se lee la cantidad de estudiantes se tiene el valor de $n$. Se usa un ciclo Para, en el que internamente se lee la nota de cada estudiante y mediante un Dependiendo De de acuerdo con la nota, se incrementa alguno de los contadores, $c1$, $c2$, $c3$, $c4$ o $c5$. En caso de ser una nota incorrecta, se muestra un mensaje y se retrocede en uno el valor de $j$; de esta forma, el ciclo Para terminará solamente si se digitaron $n$ notas correctas. Por último, se deben mostrar las estadísticas.

```
Proc: Distribución estadística de las notas
| Entero: c1, c2, c3, c4, c5, n, j, nota
| c1 ← 0, c2 ← 0, c3 ← 0, c4 ← 0, c5 ← 0
| Escriba "Por favor, digite el número total de estudiantes"
| Lea n
| Para(j = 1, n, 1)Haga
| | Escriba "Digite la nota número", j
| | Lea nota
| | DD(nota)Haga
| | | Opción 1 :
| | | | c1 ← c1 + 1
| | | Fin
| | | Opción 2 :
| | | | c2 ← c2 + 1
| | | Fin
| | | Opción 3 :
| | | | c3 ← c3 + 1
| | | Fin
| | | Opción 4 :
| | | | c4 ← c4 + 1
| | | Fin
| | | Opción 5 :
| | | | c5 ← c5 + 1
| | | Fin
| | | Sino :
| | | | Escriba "Nota incorrecta, reescríbala "
| | | | j ← j − 1
| | | Fin
| | Fin_DD
| Fin_Para
| Escriba c1, "Estudiantes sacaron 1"
| Escriba c2, "Estudiantes sacaron 2"
| Escriba c3, "Estudiantes sacaron 3"
| Escriba c4, "Estudiantes sacaron 4"
| Escriba c5, "Estudiantes sacaron 5"
Fin_proc
```

Ejemplo 3.124 Menor número divisible por todos los números del 1 al 20

2520 es el menor número que es divisible por cada uno de los números del 1 al 10 sin ningún resto. ¿Cuál es el menor número que es divisible por todos los números del 1 al 20?

*Análisis*: Eventualmente, el menor número corresponderá al mínimo común múltiplo de los primeros 20 números naturales. A continuación se muestran

tres soluciones. La primera presenta el simple hecho de ir recorriendo todos los números naturales, uno por uno, hasta encontrar el primero que sea divisible por todos los números del 1 al 20. La segunda solución calcula el mínimo común múltiplo de los primeros 20 enteros positivos, la cual se extiende a calcular el mínimo común múltiplo de los primeros $n$ enteros positivos.

En el caso de la primera solución no hace falta preguntar si el número es divisible por 1, 2, 3,4, 5, 6, 7, 8, 9, 10, pues lo será si es divisible por 11, 12, 13, 14, 15, 16, 17, 18, 19, 20; por ejemplo, si es divisible por 9, debe serlo por 18, entonces mejor se pregunta por el que excluye al otro, para no preguntar doble, es decir, por 18.

```
Proc: Solución 1
  Booleano: sw
  Entero: i
  sw ← falso
  i ← 0
  Mq(sw = falso)Haga
    i ← i + 1
    // Los puntos suspensivos significa que se pregunta si
    // el número es divisible por 20, 19, 18, 17, ..., 10 y 11
    Si(i mod 20 = 0 y ··· y i mod 11 = 0)Entonces
      sw ← verdadero
    F.Si
  Fin_Mq
  Escriba "Respuesta: ", i
Fin_proc
```

Sin embargo, si se desea preguntar si el número es divisible por los primeros $n$ enteros positivos, no se puede usar esta solución; por lo tanto, se usará un ciclo Mientras que para realizar todas estas preguntas. Si alguna resulta ser falsa, es decir, si el número resulta no ser divisible por alguno de los primeros $n$ enteros positivos, entonces el ciclo termina. Si el ciclo termina, el ciclo externo avanza al siguiente número.

```
Proc
  Entero: n, sw1, i, sw2, j
  Escriba "Ingrese el valor de n: "
  Lea n
  sw1 ← 0, i ← 0
  // Este ciclo recorrerá todos los enteros positivos
  // en búsqueda del primer entero positivo que sea divisible
  // por los primeros n enteros positivos
  Mq(sw1 = 0)Haga
    i ← i + 1
1 2
```

```
1 2
| | sw2 ← 1
| | // Se inicia en 2, puesto que todos los números son divisibles por 1
| | j ← 2
| | Mq(sw2 = 1 y j ≤ n)Haga
| | | Si(i mod j ≠ 0)Entonces
| | | | // Si no es divisible, entonces obliga a terminar
| | | | // el ciclo interior y dar paso al ciclo externo en búsqueda
| | | | // de otro número
| | | | sw2 ← 0
| | | F.Si
| | | j ← j + 1
| | Fin_Mq
| | Si(sw2 = 1)Entonces
| | | sw1 ← 1
| | F.Si
| Fin_Mq
| Escriba "Respuesta: ", i
Fin_proc
```

La tercera solución no requiere crear un algoritmo para resolver el problema inicial (los primeros 20 enteros positivos). Esta solución trata de cómo hallar el máximo y mínimo común múltiplo de 20 números, aunque, por lo visto anteriormente, no hace falta, sino hallar el mínimo común múltiplo de los 10 más grandes. Primero se deben descomponer estos números en sus factores primos

$$\begin{aligned}
11 &= 11^1\\
12 &= 2^2 \times 3^1\\
13 &= 13^1\\
14 &= 2^1 \times 7^1\\
15 &= 3^1 \times 5^1\\
16 &= 2^4\\
17 &= 17^1\\
18 &= 2^1 \times 3^2\\
19 &= 19^1\\
20 &= 2^2 \times 5
\end{aligned}$$

El mínimo común múltiplo debe ser múltiplo de 2; en particular debe ser múltiplo de la potencia de 2 más grande encontrada, es decir, debe ser múltiplo de $2^{\max(2,1,4,1,2)} = 2^4$. De la misma forma, debe ser múltiplo de la potencia más grande de 3, es decir, de $3^2$. Siguiendo este procedimiento, el mínimo común múltiplo resulta ser

$$2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 = 232792560.$$

Para extender esta solución, observe que lo que se tiene que hacer es tomar

todos los primos menores que $n$ y encontrar la potencia más alta menor que $n$ e ir acumulando este producto. Para ello se necesitará verificar si un número es primo o no. La variable $mcm$ contendrá el mínimo común múltiplo y será inicializada en 1.

```
Proc
  Entero: n, mcm, i, j, sw, prod
  Escriba "Ingrese el valor de n: "
  Lea n
  mcm ← 1
  Para(i = 2, n, 1)Haga
    // Se revisará si i es primo
    j ← 2, sw ← 0
    Mq(j ≤ √i y sw = 0)Haga
      Si(i mod j = 0)Entonces
        sw ← 1
      F.Si
      j ← j + 1
    Fin_Mq

    // Si sw = 0, entonces i es primo
    Si(sw = 0)Entonces
      prod ← 1
      Mq(prod < n)Haga
        prod ← prod * i
      Fin_Mq
      // Se asigna prod/i porque cuando el ciclo
      // Mientras que termina es porque prod es mayor que n,
      // es decir, prod está en la menor potencia mayor que n
      // y la que se necesita en realidad es la anterior
      // Por ejemplo, si i = 3 y n = 10, entonces cuando prod
      // sale del ciclo Mientras que, prod = 27, pero en realidad
      // se desea prod/i = 9
      mcm ← mcm * prod/i
    F.Si
  Fin_Para
  Escriba "El menor número que es divisible por"
  Escriba "los primeros n enteros positivos es: ", mcm
Fin_proc
```

Por último, existe una forma más eficiente para la solución de este algoritmo: la Criba de Erastótenes, sin embargo, requiere el uso de estructuras más complejas. Dicha solución se mostrará después.

Ejemplo 3.125 Suma de los múltiplos de 3 o 5

Si se listan todos los números naturales menores que 10 que son múltiplos de 3 o 5, se obtiene 3, 5, 6 y 9. La suma de estos múltiplos es 23. Encuentre la suma de todos los múltiplos de 3 o 5 menores que 1000.

*Análisis*: Se debe usar un acumulador *suma* para ir sumando todos los múltiplos de 3 o 5. Se debe usar un ciclo Para que revisará todos los números menores que 1000, por lo tanto, irá desde 1 hasta 999 con incremento de 1. Por último, se debe agregar una condición para revisar si el número es divisible por 3 o 5.

**Proc**
  **Entero:** *suma*, *i*
  $suma \leftarrow 0$
  **Para**$(i = 1, 999, 1)$**Haga**
    **Si**$(i \textbf{ mod } 3 = 0 \text{ o } i \textbf{ mod } 5 = 0)$**Entonces**
      $suma \leftarrow suma + i$
    **F.Si**
  **Fin_Para**
  **Escriba** "Respuesta: ", *suma*
**Fin_proc**

**Solución:** 233168.

Ejemplo 3.126 Número de cifras y la suma de las mismas

Realice un algoritmo que lea por pantalla un número entero y diga el número de dígitos que tiene y la suma de estos.

3621 $\Longrightarrow$ **[A]** $\Longrightarrow$ Tiene 4 dígitos y la suma de estos es 12

*Análisis*: La variable $n$ almacenará el número que se va a descomponer. Se hace una copia de $n$ en $cn$. Si el número es negativo, entonces se lo multiplica por $-1$ para trabajar con su valor absoluto. La variable $sdn$ contiene la suma de los dígitos de $n$; $nd$ contiene el número de dígitos de $n$. Para extraer las cifras del número se hace uso de las funciones **mod** y /. / realiza una división entera, por lo tanto, si se hace /10, se pueden descartar tantas cifras como se desee. Por ejemplo, $12342/10 = 1234$, $1234/10 = 123$, y así sucesivamente. Además, **mod** 10 es el resto que deja un número al ser dividido por 10; esto es equivalente a extraer su última cifra (¿por qué?). Por ejemplo, $12342 \textbf{ mod } 10 = 2$, $1234 \textbf{ mod } 10 = 4$, y así sucesivamente. El uso combinado de estas dos funciones extrae cada cifra.

**Proc**: Suma de las cifras de un número
  **Entero:** $sdn$, $nd$, $n$, $cn$
  $sdn \leftarrow 0, nd \leftarrow 0$
  **Escriba** “Por favor, digite un número entero”
  **Lea** $n$
  $cn \leftarrow n$
  **Si**$(cn < 0)$**Entonces**
    $cn \leftarrow -cn$
  **F.Si**
  **Si**$(cn = 0)$**Entonces**
    // Si el número es 0, entonces $n$ tiene 1 dígito
    // y la suma de sus cifras es 0
    $sdn \leftarrow 0$
    $nd \leftarrow 1$
  **Sino**
    **Mq**$(cn > 0)$**Haga**
      // Se acumula la suma de las cifras
      $sdn \leftarrow sdn + (cn \textbf{ mod } 10)$
      $nd \leftarrow nd + 1$
      // Se descarta la cifra sumada
      $cn \leftarrow cn/10$
    **Fin_Mq**
  **F.Si**
  **Escriba** $n$, “tiene”, $nd$, “dígitos y la suma de los dígitos de”, $n$, “es:”, $sdn$
**Fin_proc**

---

Ejemplo 3.127 Figura especial

Visualice en pantalla una figura similar a la siguiente; el usuario debe ingresar el número de líneas que se va a mostrar.

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
```

*Análisis*: La variable $n$ contiene el número de líneas que se debe mostrar. Se requiere de dos ciclos, uno que controle el salto de línea y otro que controle el número de asteriscos que se va a mostrar. Observe que si se está en la segunda línea, entonces el ciclo interno debe mostrar 2 asteriscos. Por lo tanto, si se está en la $i$-ésima línea, el ciclo interno debe mostrar $i$ asteriscos. Lo anterior quiere decir que si el ciclo externo va con $i$ desde 1 hasta $n$, entonces el ciclo interior va con $j$ desde 1 hasta $i$. Por último, en

el interior de cada ciclo Para se usa la instrucción Escriba para mostrar el asterisco.

```
Proc
| Entero: n, i, j
| Escriba "Ingrese el número n: "
| Lea n
| Para(i = 1, n, 1)Haga
| | Para(j = 1, i, 1)Haga
| | | Escriba "*"
| | Fin_Para
| Fin_Para
Fin_proc
```

# 3.10 Ejercicios propuestos

## 3.10.1 Condicionales

1. Escriba instrucciones **Si-Sino** apropiadas para cada una de las siguientes condiciones:

   *a*) Si un ángulo es igual a 90 grados, imprima el mensaje "El ángulo es un ángulo recto"; de lo contrario imprima el mensaje "El ángulo no es un ángulo recto".

   *b*) Si la temperatura está por encima de 100 grados, despliegue el mensaje "arriba del punto de ebullición del agua"; de lo contrario despliegue el mensaje "abajo del punto de ebullición del agua".

   *c*) Si el número es positivo, sume al número *sumpos*, sino sume el número a *sumneg*.

   *d*) Si la pendiente es menor que 0.5, fije la variable $flag$ en 0; de lo contrario fijar $flag$ en 1.

   *e*) Si la edad es mayor que 18, muestre el mensaje "La persona es mayor de edad"; en caso contrario, muestre "La persona es menor de edad".

   *f*) Si el ingreso es 'F', muestre el mensaje "Es una mujer"; en caso contrario, "Es un hombre".

   *g*) Si la diferencia entre $voltios1$ y $voltios2$ es menor que 0.001, fije la variable $aprox$ en 0, de lo contrario calcule $aprox$ como la cantidad $(voltios1 - voltios2)/2.0$.

   *h*) Si la frecuencia es superior a 60, despliegue el mensaje "La frecuencia es demasiado alta".

*i*) Si la diferencia entre $temp1$ y $temp2$ excede 2.4, calcule *error* como $(temp1 - temp2) * 0.01$.

*j*) Si la diferencia entre *teorico* y *experimental* excede en 5, calcule *error* como $(teorico - experimental) * 100/teorico$.

*k*) Si $x$ es mayor que $y$ y $z$ es menor que 20, lea un valor para $p$.

*l*) Si la distancia es mayor que 20 y es menor que 35, lea un valor para tiempo.

2. Haga un algoritmo que dada una fecha entre 01/01/1990 al 31/12/2010, determine si es correcta o incorrecta. Tenga en cuenta los años bisiestos.
   **Entrada:** El día, mes y año de una fecha.
   **Salida:** Deberá decir si la fecha ingresada es válida o no.

3. Haga un algoritmo que dado dos números determine si uno es divisor del otro.
   **Entrada:** Dos números enteros.
   **Salida:** Si el número menor es o no divisor del otro.

4. Escriba un algoritmo para calcular el valor de la presión en libras por pulgada cuadrada (psi) de una onda descrita como sigue:

   - Para tiempo, $t$, igual a o menor que 35 segundos, la presión es $0.46t$ psi, y para tiempo mayor que 35 segundos la presión es $0.19t + 9.45$ psi.

   - El programa deberá pedir el tiempo como entrada y deberá desplegar la presión como salida.

   **Entrada:** El tiempo
   **Salida:** Presión en psi.

5. Construya un algoritmo que reciba como datos de entrada tres números enteros, y regrese como dato de salida un mensaje que diga si esos tres números enteros pueden ser las medidas de los lados de un triángulo rectángulo.
   **Entrada:** Tres números enteros.
   Ej: 4, 4, 4.
   **Salida:** No pueden ser los lados de un triángulos rectángulo.

6. Diseñe un algoritmo en el que dado un tiempo en minutos, calcule los días, horas y minutos que le corresponden.
   **Entrada:** Tiempo en minutos (número entero).
   **Salida:** Días, horas y minutos a los que equivale.

7. Escriba un algoritmo que le pida al usuario que introduzca dos números. Si el primer número introducido es mayor que el segundo, el programa deberá imprimir el mensaje "El primer número es mayor"; de lo contrario deberá imprimir el mensaje "El primer número es menor". Pruebe su algoritmo introduciendo los números 5 y 8 y luego usando los números 11 y 2. ¿Qué piensa que desplegará su algoritmo si los dos números introducidos son iguales? Pruebe este caso.

8. La tolerancia de componentes críticos en un sistema se determina por la aplicación de acuerdo con la siguiente tabla:

| **Estado de la especificación** | **Tolerancia** |
|---|---|
| Exploración espacial | Menor que 0.1 % |
| Grado militar | Mayor que o igual a 0.1 % y menor que 1 % |
| Grado comercial | Mayor que o igual a 1 % y menor que 10 % |
| Grado de juguete | Mayor que o igual a 10 % |

Usando esta información escriba un algoritmo que acepte la lectura de tolerancia de un componente y determine la especificación que debería asignarse al componente.
**Entrada:** 0.02
**Salida:** Exploración espacial

9. Escriba un algoritmo que acepte un número, luego una letra. Si la letra que sigue al número es F, el programa tratará al número introducido como una temperatura en grados Fahrenheit, convertirá el número a los grados Celsius equivalentes y desplegará un mensaje adecuado. Si la letra que sigue al número es C, el programa tratará al número introducido como una temperatura en Celsius, convertirá el número a los grados Fahrenheit equivalentes y desplegará un mensaje adecuado. Si la letra no es f ni c, el algoritmo imprimirá el mensaje que los datos introducidos son incorrectos y terminará. Use una cadena Si-Sino en su algoritmo y use las fórmulas de conversión:

$$Celsius = \frac{5}{9} \times (Fahrenheit - 32.0)$$
$$Fahrenheit = \frac{9}{5} \times (Celsius - 32.0)$$

**Entrada:** 32.0
**Entrada:** F
**Salida:** La temperatura en grados Celsius es 0.

10. Escriba un programa que lea tres enteros y emita un mensaje que indique si están o no en orden numérico.

**Entrada:** 3
**Entrada:** 2
**Entrada:** 5
**Salida:** Los números no están orden.

**Entrada:** 3
**Entrada:** 4
**Entrada:** 5
**Salida:** Los números si están orden.

11. Escriba una sentencia Si-Sino que clasifique un entero en una de las siguientes categorías y escriba un mensaje adecuado:

$$x < 0 \qquad 0 \leq x \leq 100 \qquad x > 100$$

**Entrada:** 5
**Salida:** El número es menor o igual que 100 y mayor o igual que 0.

12. Escriba un programa que introduzca el número de un mes (1 a 12) y visualice el número de días de ese mes.

    **Entrada:** 12
    **Salida:** El mes 12 tiene 31 días

13. La fuerza de atracción entre dos masas, $m_1$ y $m_2$, separadas por una distancia $d$ está dada por la fórmula

    $$F = \frac{Gm_1m_2}{d^2}$$

    Donde $G$ es la constante de gravitación universal

    $$G = 6.673 \times 10^{-8}\,\text{cm}^3/\text{g-seg}^2$$

    Escriba un algoritmo que lea la masa de dos cuerpos y la distancia entre ellos y a continuación obtenga la fuerza gravitacional entre ellos. La salida debe ser en dinas; una dina es igual a 1 g-cm/$s^2$.

    **Entrada:** Dos masas en gramos, distancia en centímetros
    **Salida:** Fuerza de atracción en dinas

    **Entrada Ejemplo:** 2400, 12000, 0.1
    **Salida Ejemplo:** 192.18 g-cm/$s^2$.

14. Escriba un algoritmo que clasifique un entero leído del teclado de acuerdo a los siguientes puntos:

    *a*) Si es 30 o mayor, o negativo, visualice un mensaje en ese sentido;

    *b*) si es un numero primo, potencia de 2 o un número compuesto, visualice el mensaje correspondiente;

    *c*) si es 0 o 1, muestre “cero” o “unidad”, según corresponda.

    **Entrada:** 12
    **Salida:** El número corresponde a la categoría B.

15. Escriba un algoritmo que reciba como entrada las coordenadas de dos vértices opuestos de un rectángulo, imprima las coordenadas de los otros dos vértices. El usuario ingresará las dos coordenadas $(x_1, y_1)$ y $(x_2, y_2)$ en el orden $x_1$,$y_1$,$x_2$,$y_2$.
    **Entrada:** $5, 2, 8, -3$
    **Salida:** $5, -3, 8, 2$

16. Se sabe que el primer día de un mes es domingo. Haga un algoritmo que reciba como entrada un número entero $n$ $(1 \leq n \leq 31)$ correspondiente a un día de ese mes e imprima en qué día de la semana cae ese día $n$.
**Entrada:** 1
**Salida:** domingo
**Entrada:** 11
**Salida:** miércoles

17. El costo de inscripción en un concurso de Algoritmia es de 100 000 pesos por inscripción de la universidad más 5 000 pesos por cada alumno participante. Si la universidad se inscribe con al menos 250 participantes, no se cobran los 100 000 pesos de inscripción del colegio. Haga un algoritmo que reciba como entrada el número de estudiantes que un colegio va a inscribir en un evento de olimpiadas y que calcule el monto del pago por la inscripción.
**Entrada:** 50
**Salida:** 350000
**Entrada:** 252
**Salida:** 1260000

18. Haga un programa que reciba como entrada tres números y diga si hay alguno que sea múltiplo de los otros dos; en caso afirmativo debe decir cuál es.
**Entrada:** 5, 3, 2
**Salida:** No
**Entrada:** 5, 20, 4
**Salida:** Si, 20

19. Haga un algoritmo que reciba como entrada las coordenadas de dos vértices opuestos de un rectángulo e imprima el área del rectángulo.
**Entrada:** 5, 2, 8, -3
**Salida:** 15

20. Dado un número entero $n$ $(1 \leq n \leq 26)$, muestre en pantalla un triángulo alfabético como el mostrado en el ejemplo. El valor ANSII que corresponde al carácter 'a' es 96.
**Entrada:** 3
**Salida:**
a
bb
ccc

21. Un archivo contiene dos fechas en el formato día (1 a 31), mes (1 a 12) y año (entero de cuatro dígitos), correspondientes a la fecha de nacimiento y la fecha actual, respectivamente. Escriba un programa que calcule y visualice la edad del individuo. Si es la fecha de un bebé (menos de un año de edad), la edad se debe dar en meses y días; en caso contrario, la edad se debe calcular en años.

### 3.10.2 Ciclos

22. Describa la salida de los siguientes ciclos:

    *a*) **Proc**
    **Entero:** $i$, $j$
    **Para**$(i = 1, 5, 1)$**Haga**
    **Escriba** $i$
    **Para**$(j = i, 1, -2)$**Haga**
    **Escriba** $j$
    **Fin_Para**
    **Fin_Para**
    **Fin_proc**

    *b*) **Proc**
    **Entero:** $i$, $j$, $k$
    **Para**$(i = 3, 1, -1)$**Haga**
    **Para**$(j = 1, i, 1)$**Haga**
    **Para**$(k = i, j, -1)$**Haga**
    **Escriba** $i$,$j$,$k$
    **Fin_Para**
    **Fin_Para**
    **Fin_Para**
    **Fin_proc**

23. Escriba instrucciones **Para** individuales para los siguientes casos:

    *a*) Al usar un contador $i$ que tiene valor inicial de 1, un valor final de 20 y un incremento de 1.

    *b*) Al usar un contador *icuenta* que tiene un valor inicial de 1, un valor final 20 y un incremento de 2.

    *c*) Al usar un contador $j$ que tiene un valor inicial de 1, un valor final de 100 y un incremento de 5.

    *d*) Al usar un contador *icuenta* que tiene valor inicial de 20, un valor final de 1 y un incremento de -1.

    *e*) Al usar un contador *icuenta* que tiene valor inicial de 20, un valor final de 1 y un incremento de -2.

    *f*) Al usar un contador *cuenta* que tiene valor inicial de 1.0, un valor final de 16.2 y un incremento de 0.2.

    *g*) Al usar un contador *xcnt* que tiene valor inicial de 20.0, un valor final de 10.0 y un incremento de -

24. Determine el número de veces que se ejecuta cada ciclo en el ejercicio anterior.

25. Determine el valor de la variable *total* después que se ejecuta cada uno de los siguientes ciclos.

a) **Proc**
  **Entero:** *total*, *i*
  $total \leftarrow 0$
  **Para**($i = 1, 10, 1$)**Haga**
    $total \leftarrow total + 1$
  **Fin_Para**
**Fin_proc**

b) **Proc**
  **Entero:** *total*, *cuenta*
  $total \leftarrow 1$
  **Para**($cuenta = 1, 10, 1$)**Haga**
    $total \leftarrow total * 2$
  **Fin_Para**
**Fin_proc**

c) **Proc**
  **Entero:** *total*, *i*
  $total \leftarrow 0$
  **Para**($i = 10, 15, 1$)**Haga**
    $total \leftarrow total + i$
  **Fin_Para**
**Fin_proc**

d) **Proc**
  **Entero:** *total*, *i*
  $total \leftarrow 50$
  **Para**($i = 1, 10, 1$)**Haga**
    $total \leftarrow total - 1$
  **Fin_Para**
**Fin_proc**

e) **Proc**
  **Entero:** *total*, *icnt*
  $total \leftarrow 1$
  **Para**($icnt = 1, 8, 1$)**Haga**
    $total \leftarrow total * icnt$
  **Fin_Para**
**Fin_proc**

f) **Proc**
  **Entero:** *total*, *j*
  $total \leftarrow 1.0$
  **Para**($j = 1, 5, 1$)**Haga**
    $total \leftarrow total/2.0$
  **Fin_Para**
**Fin_proc**

26. Determine cuál es la salida de este ciclo:

```
Proc
  Entero: i, j
  i ← 0
  Mq(i * i < 10)Haga
    j ← i
    Mq(j * j < 100)Haga
      Escriba i + j
      j ← j * 2
    Fin_Mq
    i ← i + 1
  Fin_Mq
  Escriba "Fin"
Fin_proc
```

27. Escriba un algoritmo que calcule y visualice el más grande, el más pequeño y la media de $N$ números. El valor de $N$ se solicitará al principio del algoritmo y los números serán introducidos por el usuario.

    **Entrada:** 1 2 3 2 3 5
    **Salida:** Media: 2.66, Mayor: 5, Menor: 1

28. Modifique el algoritmo anterior, de tal forma que muestre solo si hay menor absoluto, el menor, y si hay mayor absoluto, el mayor. Si un número $n$ es menor absoluto, entonces $n$ no se repite.

    **Entrada:** 1 2 3 2 3 5 5
    **Salida:** Media: 2.66, Menor absoluto: 1, No hay mayor absoluto.

29. Calcule el factorial de un número leído utilizando las sentencias Mientras que y Para.

    **Entrada:** 5
    **Salida:** 120

30. Calcule la suma de una serie de números leídos del teclado. El algoritmo debe indicar qué número interrumpe la lectura.

    **Entrada:** 1 5 3 5 0
    **Salida:** 14

31. Cuente el número de enteros negativos introducidos por el usuario. Igual al inciso anterior, el algoritmo debe indicar qué número detiene la lectura.

    **Entrada:** -1 2 -3 8 9 9 -4 0
    **Salida:** 3

32. Calcule la suma de los términos de la serie

$$\sum_{i=1}^{n} \frac{1}{2^i}$$

    El usuario debe ingresar el valor de $N$; tenga en cuenta que entre más grande sea $N$, el resultado debe acercarse a 1.

33. Haga un algoritmo para hallar de un conjunto de $N$ números qué porcentaje son cero, qué porcentaje son positivos y el porcentaje de números negativos.

    **Entrada:** -1 2 -3 8 9 9 -4 0 5 5 6 0
    **Salida:** Positivos: 58.3 %, Ceros: 16.6 %, Negativos: 25 %.

34. Haga un algoritmo para hallar cuántos números se debieron haber leído de un conjunto dado para que la suma de los negativos dé en valor absoluto mayor que 1200.

    **Entrada:** -16 2 -300 8 9 9 -4000
    **Salida:** 7

35. Diseñe un algoritmo que reciba como dato un número entero y a partir de este genere un número de un dígito (entre 0 y 9) sumando los dígitos tantas veces como sea necesario.

    **Entrada:** 3265
    **Salida:** 7

36. Construya un algoritmo que reciba como datos de entrada $n$ números enteros y regrese como dato de salida el mayor de estos números.

    **Entrada:** 4 6 2 7 3
    **Salida:** El número mayor es: 7

37. Diseñe un algoritmo que reciba como dato de entrada un número entero positivo, $n$, y regrese como dato de salida el valor de la siguiente serie

$$\frac{\pi}{4} = \sum_{i=0}^{n} \frac{(-1)^i}{(2i+1)}$$

    **Entrada:** 5
    **Salida:** 0.7440115440

38. Construya un algoritmo que reciba como dato de entrada un número entero positivo, $n$, y regrese como dato de salida la representación de este número en binario.
    **Entrada:** 10
    **Salida:** El número representado en el sistema binario es 1010

39. Construya un algoritmo que reciba como dato de entrada un número entero positivo, $n$, y regrese como dato de salida la representación de este número en hexadecimal.

    **Entrada:** 11
    **Salida:** El número representado en el sistema hexadecimal es B

40. Realice un algoritmo en el que dado $n$ números naturales, determine y escriba qué porcentaje son pares y que porcentaje son primos.

    **Entrada:** 4 5 6 2 1 -5 9 8
    **Salida:** Pares: 50 %, Primos: 25 %

41. Realice un algoritmo en el que dado un número, diga si es o no un número de Armstrong.
Un número de $n$ dígitos es un número de Armstrong si la suma de las potencias $n$-ésimas de los dígitos que lo componen es igual al mismo número.
**Entrada:** 1634
**Salida:** Sí, el número es de Armstrong.

42. Para encontrar el máximo común divisor (mcd) de dos números se emplea el algoritmo de Euclides, que se puede describir así: dados los enteros, $a$ y $b$ ($a > b$), se divide $a$ por $b$, y se obtiene el cociente $q_1$ y el resto $r_1$. Si $r_1 \neq 0$, se divide $b$ por $r_1$, y se obtiene el cociente $q_2$ y el resto $r_2$. Si $r_2 \neq 0$, se divide $r_1$ por $r_2$, y se obtiene cocientes y restos sucesivos. El proceso continúa hasta obtener un resto igual a 0. El resto anterior a este es el máximo común divisor de los números iniciales. Diseñar un algoritmo que calcule el máximo común divisor mediante el algoritmo de euclides.

43. Diseñe un algoritmo que encuentre el menor número primo ingresado por el usuario; tenga en cuenta que la lectura de números terminará con el número que el usuario ingresa. Se deben realizar verificaciones respectivas para conocer si el número es o no primo.

44. Escriba un algoritmo el cual permita calcular el coeficiente binomial con una función factorial.

$$\binom{m}{n} = \frac{m!}{n!(m-n)!} \qquad m! = \begin{cases} 1 & \text{si } m = 0 \\ 1 \times 2 \times 3 \times \cdots \times m & \text{si } m > 0 \end{cases}$$

45. Escriba una función que permita calcular la serie

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

46. Realice un algoritmo que calcule la función que corresponda para un $x$ ingresado por el usuario y el número de términos $m$ de la serie que él desee evaluar. En el caso de las funciones trigonométricas, el $x$ se encuentra en radianes.

   *a*) Función Exponencial

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \quad \text{para todo } x$$

   *b*) Función Logaritmo natural

$$\ln(1+x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} x^n \quad \text{para } |x| < 1$$

*c*) Función Seno

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \quad \text{para todo } x$$

*d*) Función Coseno

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \quad \text{para todo } x$$

*e*) Función Arcoseno

$$\arcsin x = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} \quad \text{para } |x| < 1$$

*f*) Función Arcotangente

$$\arctan x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1} \quad \text{para } |x| < 1$$

*g*) Función Seno hiperbólico

$$\sinh x = \sum_{n=0}^{\infty} \frac{1}{(2n+1)!} x^{2n+1} \quad \text{para todo } x$$

*h*) Función Coseno hiperbólico

$$\cosh x = \sum_{n=0}^{\infty} \frac{1}{(2n)!} x^{2n} \quad \text{para todo } x$$

*i*) Función Arcoseno hiperbólico

$$\sinh^{-1} x = \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} \quad \text{para } |x| < 1$$

*j*) Función Tangente hiperbólico

$$\tanh^{-1} x = \sum_{n=0}^{\infty} \frac{1}{2n+1} x^{2n+1} \quad \text{para } |x| < 1$$

*k*) Serie Geométrica

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n \quad \text{para } |x| < 1$$

47. Un modelo de población mundial, en miles de millones de personas, está dado por la ecuación $Poblacion = 6.0e^{0.02t}$, donde $t$ es el tiempo en años ($t = 0$ representa enero de 2000 y $t = 1$ representa enero de 2001). Usando esta fórmula escriba un algoritmo que despliegue una tabla de población anual para los años de enero de 2005 a enero de 2025.

48. Escriba un programa que calcule y despliegue valores para $y$ cuando

$$y = \frac{xz}{x - z}$$

Su programa deberá calcular $y$ para valores de $x$ que varían entre 1 y 5 y valores de $z$ que varían entre 2 y 6. La variable $x$ deberá controlar el ciclo exterior e incrementarse en pasos de 1 y $z$ deberá incrementarse en pasos de 2. Su algoritmo deberá desplegar también el mensaje *Función Indefinida* cuando sea necesario.

49. Los lenguajes ensambladores para algunos microprocesadores no tienen una operación de multiplicación. Aunque hay algoritmos sofisticados para llevar a cabo la multiplicación en estos casos, un método simple multiplica por adición repetida. En este caso la eficiencia del algoritmo puede incrementarse usando ciclos anidados. Por ejemplo, para multiplicar un número por doce, primero suma el número tres veces y luego suma el resultado cuatro veces. Esto solo requiere siete adiciones en vez de doce. Usando esta información escriba un algoritmo que multiplique 33, 47 y 83 por 1001 usando tres ciclos y luego despliegue el resultado. (*Sugerencia:* $1001 = 7 \times 11 \times 13$).

50. Usando una instrucción Hacer-Hasta escriba un programa para aceptar una calificación. El programa deberá solicitar una calificación en forma continua en tanto se introduzca una calificación inválida. Una calificación es inválida si es menor que 0 o mayor que 100. Después que se ha introducido una calificación válida, su programa deberá desplegar el valor de la calificación introducida. Luego:

    *a*) Modifique el algoritmo anterior de modo que el usuario sea alertado cuando se ha introducido una calificación inválida. Además, se debe permitir al usuario salir del programa introduciendo el número 999.

    *b*) Modifique de modo que termine en forma automática después que se han introducido cinco calificaciones inválidas.

51. Escriba un algoritmo que solicite en forma continua que se introduzca una calificación. Si la calificación es menor que 0 o mayor que 100, su algoritmo deberá imprimir un mensaje apropiado que informe al usuario que se ha introducido una calificación inválida; de lo contrario, la calificación deberá sumarse a un total. Cuando se introduzca una calificación de 999, el algoritmo deberá salir del ciclo de repetición y calcular y desplegar el promedio de las calificaciones válidas introducidas.

52. Escriba un algoritmo para invertir los dígitos de un número entero positivo. Por ejemplo, si se introduce el número 8735, el número desplegado deberá ser 5378. (*Sugerencia:* use una instrucción Hacer-Hasta y continuamente quite y despliegue el dígito de las unidades del número). Si la variable *num* en un inicio contiene el número introducido, el dígito de las unidades se obtiene como $num\,MOD\,10$. Después que se despliega un dígito de las unidades, dividir el número entre 10 establece el número para la siguiente iteracción. Por tanto, $8735\,MOD\,10$ es 5 y $8735\,DIV\,10$ es 873. La instrucción Hacer-Hasta deberá continuar en tanto el número restante no sea cero.

53. Dado un número $n$, y una aproximación para su raíz cuadrada, puede obtenerse una aproximación más cercana a la raíz cuadrada real usando la fórmula

$$\text{aproximación previa} = \frac{\frac{n}{\text{aproximación previa}} + \text{aproximación previa}}{2}$$

Usando esta información escriba un algoritmo que indique al usuario que introduzca un número y una estimación inicial de su raíz cuadrada. Usando estos datos de entrada, su programa deberá calcular una aproximación a la raíz cuadrada que tenga una precisión hasta 0.0001. (*Sugerencia:* detenga el ciclo cuando la diferencia entre dos aproximaciones sea menor que 0.0001).

54. El método de Newton-Raphson puede utilizarse para encontrar las raíces de cualquier ecuación $y(x) = 0$. En este método, la $(i+1)$-ésima aproximación, $x_{i+1}$, a una raíz de $y(x) = 0$ está dada en términos de la $i$-ésima aproximación, $x_i$, por la fórmula

$$x_{i+1} = x_i - \frac{y(x_i)}{y'(x_i)}$$

Por ejemplo, si $y(x) = 3x^2 + 2x - 2$, entonces $y'(x) = 6x + 2$, y las raíces se encuentran haciendo una estimación razonable para una primera aproximación, $x_i$, y haciendo iteraciones usando la ecuación

$$x_{i+1} = x_i - \frac{3{x_i}^2 + 2x_i - 2}{6x_i + 2}$$

*a*) Usando el método de Newton-Raphson encuentre las dos raíces de la ecuación $3x^2 + 2x - 2 = 0$. (*Sugerencia:* hay una raíz positiva y una raíz negativa).

*b*) Extienda el algoritmo escrito para el inciso anterior de modo que encuentre las raíces de cualquier función cuadrática cuando los coeficientes $a, b, c$ de $ax^2 + bx + c$ son introducidos por el usuario.

55. Una serie aritmética se define por

$$a + (a + d) + (a + 2d) + (a + 3d) + \cdots + [(a + (n-1)d)]$$

donde $a$ es el primer término, $d$ es la "diferencia común" y $n$ es el número de términos que se van a sumar. Usando esta información escriba un algoritmo

que use el ciclo Mientras que para desplegar cada término y para determinar la suma de la serie aritmética si se tiene que $a$, $d$ y $n$ son introducidos por el usuario. Asegúrese de que su programa despliegue el valor calculado.

**Entrada:** $a = 1, d = 1, n = 3$
**Salida:** 6

56. Una serie geométrica se define por

$$a + ar + ar^2 + ar^3 + ? + ar^{n-1}$$

donde $a$ es el primer término, $r$ es la "razón común" y $n$ es el número de términos en la serie. Usando esta información escriba un algoritmo que utilice un ciclo Mientras que para desplegar cada término y para determinar la suma de una serie geométrica si se tiene que $a$, $r$ y $n$ son introducidos por el usuario. Asegúrese de que su programa despliega el valor que se ha calculado.

**Entrada:** $a = 1, r = 2, n = 3$
**Salida:** 7

57. Además del promedio aritmético de un conjunto de números se puede calcular una media aritmética y una media armónica. La media aritmética de un conjunto de n números $x_1, x_2, x_3, \cdots, x_n$ se define como

$$\sqrt[n]{\prod_{i=1}^{n} x_i} = \sqrt[n]{x_1 x_2 \cdots x_n}$$

Y la media armónica como

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}}$$

Usando estas fórmulas escriba un algoritmo que continúe aceptando números hasta que se introduzca el numero 999, y luego calcule y despliegue tanto la media aritmética como la armónica de los números introducidos, excepto el 999. (*Sugerencia:* será necesario que su programa cuente en forma correcta el número de valores introducidos.)

58. Escriba un programa que reciba como entrada dos enteros positivos, $a$ y $b$, con $a \leq b$ e imprima todos los cuadrados perfectos que están entre $a$ y $b$ (incluyendo $a$ y $b$). Un número es cuadrado perfecto si es el cuadrado de algún número entero, por ejemplo, 9 es cuadrado perfecto ($9 = 3^2$).
**Entrada:** 8 16
**Salida:** 9 16

59. Haga un algoritmo que reciba como entrada dos números enteros, $a$ y $b$, e imprima la suma de todos los números entre $a$ y $b$ (incluyendo $a$ y $b$). Se garantiza que $a \leq b$.
**Entrada:** 1, 5
**Salida:** 15

60. Los siguientes datos se recolectaron en un viaje frecuente en automóvil:

| Millas | Galones |
|---|---|
| 22495 | Tanque lleno |
| 22841 | 12.2 |
| 23185 | 11.3 |
| 23400 | 10.5 |
| 23772 | 11.0 |
| 24055 | 12.2 |
| 24434 | 14.7 |
| 24804 | 14.3 |
| 25276 | 15.2 |

Escriba un algoritmo que acepte un valor de millas y galones y calcule las millas por galón (mpg) logradas para ese segmento del viaje. Las millas por galón se obtienen como la diferencia en millas entre llenadas del tanque divido entre el número de galones de gasolina utilizados desde que se llenó el tanque.

61. Modifique el algoritmo escrito en el inciso anterior para calcular y desplegar adicionalmente las mpg acumuladas después de cada llenada de tanque. Las mpg acumuladas se calculan como la diferencia entre el millas en cada llenada y el millas al principio del viaje dividido entre la suma de los galones usados hasta ese punto en el viaje.

62. Haga un algoritmo que reciba como entrada un entero positivo $n$ $(1 \leq n \leq 20)$, a continuación $n$ enteros y que diga de esos $n$ enteros cuántos son impares.
**Entrada:** 5, 4, 3, 8, 1, 24
**Salida:** 3

63. Haga un algoritmo que reciba como entrada un número entero positivo y que imprima los factores primos de dicho número.
**Entrada:** 60
**Salida:** 2, 3, 5

64. Una máquina comprada por 28 000 pesos se devalúa 4000 pesos por año durante siete años. Escriba un algoritmo que calcule y despliegue una tabla de devaluación para siete años. La tabla deberá tener la forma

| **Año** | **Devaluación** | **Valor al final del año** | **Devaluación acumulada** |
|---|---|---|---|
| 1 | 4000 | 24000 | 4000 |
| 2 | 4000 | 20000 | 8000 |
| 3 | 4000 | 16000 | 12000 |
| 4 | 4000 | 12000 | 16000 |
| 5 | 4000 | 8000 | 20000 |
| 6 | 4000 | 4000 | 24000 |
| 7 | 4000 | 0 | 28000 |

65. Determine el número de billetes y monedas de curso legal equivalentes a cierta cantidad de pesos (Cambio óptimo).
**Entrada:** 32300
**Salida:**
Billetes de 20 mil: 1
Billetes de 10 mil: 1
Billetes de 2 mil: 1
Monedas de 200: 1
Monedas de 100: 1

66. Un automóvil viaja a una velocidad promedio de 55 millas por hora durante cuatro horas. Escriba un algoritmo que despliegue la distancia, en millas, que el automóvil ha recorrido después de 1, 2 o varias horas hasta el final del viaje.

### 3.10.3 Repaso

67. Dado una base $k$ y un número $n$, diseñe un algoritmo que determine el número de ceros en los que termina $n!$ y el número de cifras cuando es representado en dicha base. Se garantiza la existencia de los símbolos $0, b_1, b_2, \ldots, b_{k-1}$ para los números $0, 1, 2, \ldots, k-1$, respectivamente.

    Supongamos que $k = 4$, luego los números que representan a cualquier número son 0123. Por otro lado, supongamos $n = 8$, luego $n! = 40320$, que en base 4 sería $21312000_4$, luego el número termina en 3 ceros y tiene 8 cifras.

    **Entrada:** $n, k$
    **Salida:** Número de cifras de $n!$ y número de ceros en los que termina, en base $k$.

    **Entrada Ejemplo:** $n = 8, k = 4$
    **Salida Ejemplo:** Tiene 8 cifras y termina en 3 ceros.

68. Modifique el algoritmo anterior de tal forma que no sea necesario evaluar $n!$. Piense en que si $n = 2012$, ¿cuántas cifras se necesitan para calcular inicialmente $n!$?, ¿qué tamaño se requiere en el computador? Escriba su algoritmo en C++, Java o algún lenguaje de programación. Compare el tiempo de ejecución de la versión que calcula $n!$ con la versión que no lo hace. Saque sus conclusiones.

69. La fracción 49/98 es una fracción curiosa; un matemático inexperto en el intento de simplificar creyó incorrectamente que $49/98 = 4/8$, sin embargo, resultó ser correcto, al cancelar los 9's. Se consideran ejemplos triviales fracciones como $30/50 = 3/5$ (en los que se cancelan los ceros). Hay exactamente cuatro ejemplos no triviales de este tipo de fracción, menor que uno en valor, y que contienen dos dígitos en el numerador y el denominador. Si el producto

de estas cuatro fracciones forma una fracción $a/b$, en la que $a$ y $b$ no tienen factores en común, encuentre el valor de $b$.
**Solución**: 100.

## 3.11 Conclusiones

El aprendizaje y la práctica de las estructuras lógicas básicas para la construcción de algoritmos permiten:

- Organizar las acciones de entrada de datos, procesos aritmético-lógicos o de asignación, y salida de informaciones derivadas del correcto funcionamiento del algoritmo.

- Las estructuras lógicas algorítmicas condicionales simples y compuestas, unidas a las anteriores, permiten hacer algoritmos que necesitan acciones de decisión disyuntas (Sí o No); útiles para la simulación de procesos de árboles de decisión; clasificaciones de grupos, teoría de juegos, simulaciones y procesos estocásticos, entre otros, en los cuales se tenga que tomar una ruta de acción sujeta a una decisión.

- La acción de decisión en la lógica humana, y representada en algoritmos por las condicionales expuestas, se complementa con la estructura también condicional de lógica Dependiendo De; dicha condición sirve cuando se requiere, entre varias acciones, seleccionar una sola, tal que al operar la acción seleccionada de una forma continuada se ejecuten varias estructuras algorítmicas que cumplan con la función de la acción seleccionada y se termine la lógica del algoritmo; estructuras que necesariamente son disyuntas a las acciones que no fueron seleccionadas dentro del algoritmo.

- Las estructuras algorítmicas repetitivas representadas por el Para, el Mientras que y el Haga Hasta, complementadas con las anteriores, apoyan la fabricación de algoritmos que necesitan ciclos, procesos repetitivos, acciones paso a paso, procesos de principio a fin o de fin a principio de un fenómeno, procesos circulares, procesos encapsulados al anidar las estructuras repetitivas o acciones oscilantes.

- Con base en las estructuras algorítmicas expuestas en este capítulo se obtienen diseños algorítmicos útiles que, como piezas de software funcionales, se pueden integrar a nivel de subsistemas de algoritmos iniciales en piezas de software más complejas.

- La potencia y el alcance de las estructuras algorítmicas expuestas solo dependen del ingenio y creatividad con que el diseñador de algoritmos las pueda utilizar en la resolución de un problema de la vida real.

- Es importante resaltar que los casos presentados a nivel de resolución de problemas están estrechamente relacionados con sistemas de información reales con relación a sistemas universitarios, bancarios o de administración de edificios; y fueron desarrollados todos ellos con la integración de las estructuras lógicas básicas para la construcción de algoritmos; se visiona, por lo tanto, su importancia y el horizonte de su alcance en la producción de software. El paso siguiente es complementar la estructuras lógicas algorítmicas básicas con las estructuras de datos y su control; tema que será desarrollado en el siguiente capítulo.

# Bibliografía

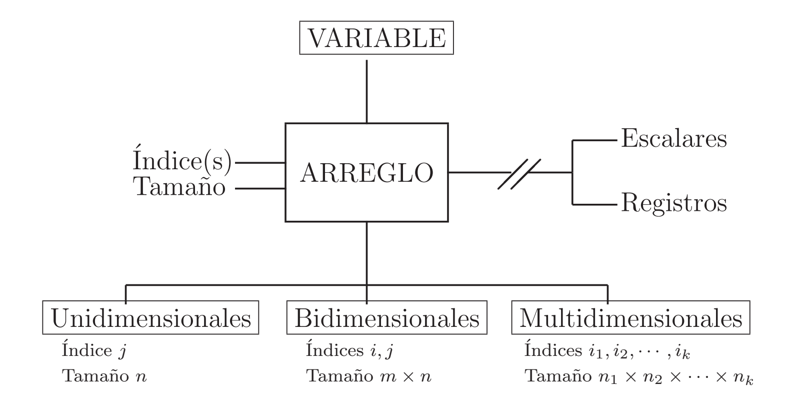
Bronson, G. J. & Borse, G. J. (2010). *C++ for engineers and scientists.* Boston: Thomson Course Technology.

García, L. (2010). *Todo lo básico que debería saber sobre Programación Orientada a Objetos en Java.* Barranquilla: Ediciones Uninorte.

Rueda, F. (1981). *Curso de Estructuras de Información*, vol. I. Bogotá, D.C.: Publicaciones de la Facultad de Ingeniería, Universidad de los Andes.

# Capítulo 4

# Estructuras de datos




Un *arreglo* es una variable en la que se almacena, en forma secuencial, un conjunto de datos del mismo tipo a partir de una posición inicial. Como toda variable, un arreglo tiene las siguientes características:

- Nombre
- Tipo
- Dirección
- Contenido

Pero tiene unos atributos adicionales:

- Tamaño, el cual indica el número de elementos que se almacenan en el arreglo.

- Índice, es un puntero que permite acceder a un elemento específico del arreglo.

Todos los arreglos se almacenan internamente, en la memoria del computador, de manera secuencial, como se ilustra a continuación:

$A:$

| $a_1$ | $a_2$ | ... | $a_{i-1}$ | $a_i$ | $a_{i+1}$ | ... | $a_{k-1}$ | $a_k$ |
|---|---|---|---|---|---|---|---|---|

Sin embargo, el programador puede tener distintas vistas lógicas para la manipulación del mismo conjunto de datos. Estas vistas corresponden a lo que se denomina la *dimensión*; dependiendo de la manera como los datos sean estructurados, los arreglos resultarán ser de una, dos o más dimensiones.

Según su dimensión, los arreglos se clasifican en:

- Unidimensionales
- Bidimensionales
- Multidimensionales

# 4.1 Concepto de vector unidimensional

Un *vector* es una estructura de datos compuesta por un conjunto de elementos de la misma especie, los cuales son direccionados por un único subíndice, el cual está organizado linealmente, con el fin de almacenar un arreglo de informaciones que se disponen secuencialmente de una forma contigua dentro del vector. Por su organización, el vector recibe también el nombre de **arreglo unidimensional**.

El vector está compuesto por cuatro elementos:

- El nombre del vector, que identifica la estructura de datos llamada vector, arreglo lineal o unidimensional.
- El subíndice, que direcciona los contenidos de información del vector.
- Las informaciones contenidas en las direcciones ubicadas por los subíndices del vector.

- El rango de almacenamiento del vector, desde un punto inicial hasta un punto final o tope del vector.

Los vectores también reciben el nombre de arreglos por contener elementos de una misma especie; luego "... los arreglos ocupan espacios de memoria" (Deitel & Deitel, 2004, p. 255); espacios de memoria que se direccionan con índices, como se muestra en el esquema.

Esquemáticamente, el vector se representa en la figura 4.1.

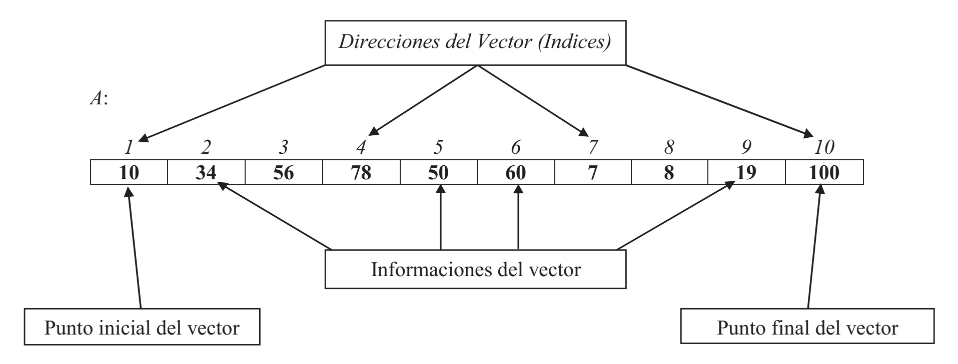



*Figura 4.1* Representación esquemática de un vector

El nombre del vector es $A$, lo que identifica la totalidad de la estructura como un arreglo unidimensional.

Las direcciones, o posiciones, del vector son subíndices, para este caso dispuestos de una forma lineal, o sea, unos a continuación de otros, denotados por 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 para este caso. Los subíndices son **totalmente diferentes** a los contenidos de informaciones, los cuales se identifican así: $A[1] = 10$; en la primera (1) dirección del vector se encuentra la información 10, en tanto que en la quinta (5) dirección del vector se encuentra la información 50. Se recalca que en la dirección siete (7) del vector o $A[7]$ se encuentra la información 7, pero su semántica es diferente, porque un concepto es el *subíndice* y otro concepto totalmente diferente es la **información** que direcciona el subíndice del vector.

El rango de almacenamiento del vector $A$ parte de una posición inicial subíndice uno (1) y llega una posición de dirección final (10).

Los subíndices dispuestos de una forma lineal 1, 2, 3, . . ., 10, permiten la organización en secuencia de las informaciones contenidas en el vector $A$; secuencia de informaciones que conducen a identificar que en la organización presentada la información 78 precede a la información 50, pero la información 78 es sucesora de la información 56; por esta razón, si 4 es el subíndice del vector en análisis, entonces $A[4] = 78$, y el subíndice sucesor de 4 es el 5, luego $A[5] = 50$, de lo cual se puede inferir correctamente que 78 precede a 50; ahora bien, estando en $A[4] = 78$,

el subíndice predecesor de 4 es 3 y $A[3] = 56$, de lo cual se puede concluir que la información 78 es sucesora de la información 56 para este caso.

Se identifica el vector en su conjunto como estructura organizada por su nombre ($A$), en tanto que las informaciones contenidas en el vector se denotan por $A[i]$, variando el subíndice $i$ en el rango desde el valor 1 hasta el valor 10 para este caso.

Luego con base en la identidad del vector y teniendo en cuenta su subíndice asociado, las estructuras de control lógicas **Mientras que, Para** o **Haga Hasta** se utilizan para controlar los subíndices del vector desde un punto inicial hasta un punto final del contenido de almacenamiento del vector.

Ejemplo 4.1 Impresor de elementos contenidos en un vector

Construya un algoritmo para leer e imprimir los $n$ elementos de un vector ($n$ leído) que contiene números enteros positivos.
*Análisis*: Supuesto $n$ el número de elementos del vector igual a 10, el vector va a contener números enteros positivos, dispuestos esquemáticamente en la figura 4.2.

$A$:

| *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
|---|---|---|---|---|---|---|---|---|---|
| **22** | **34** | **56** | **57** | **68** | **70** | **71** | **87** | **88** | **99** |

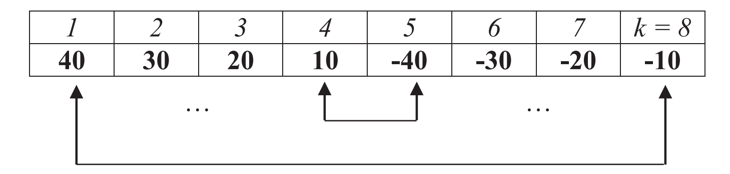
*Figura 4.2* Vector de $n$ enteros positivos

Con base en la figura 4.2, para leer los elementos del vector $A$ se debe tener un subíndice que varíe desde el punto inicial 1 hasta el punto final 10, lo cual se puede realizar con una estructura lógica de control Para y en su ciclo Leer $A[i]$ para $1 \leq i \leq 10$.

**Proc**: Lectura e impresión de un vector $A$ de $n$ elementos
  **Entero:** $n$, $i$, $A[50]$
  **Lea** $n$ // Leer el número de elementos del vector $A$
  // Ciclo repetitivo que lee los elementos del vector
  **Para**$(i = 1, n, 1)$**Haga**
    // Leer los elementos del vector identificado como $A$
    **Lea** $A[i]$
  **Fin_Para**
  // Ciclo repetitivo que escribe los elementos del vector
  **Para**$(i = 1, n, 1)$**Haga**
    // Escribe el elemento $i$-ésimo del vector $A$
    **Escriba** $A[i]$
  **Fin_Para**
**Fin_proc**

Ejemplo 4.2 Intercambiador de los elementos de un vector

Suponga un vector $V$ de $k$ elementos que contiene números enteros en sus unidades de información. Diseñe un algoritmo para intercambiar el primer elemento con el último elemento del vector, el segundo con el antepenúltimo elemento del vector, y así sucesivamente hasta alcanzar el punto medio del vector o arreglo unidimensional.
*Análisis*: Suponga un vector de $k = 8$ elementos, presentado en la figura 4.3.



| *1* | *2* | *3* | *4* | *5* | *6* | *7* | *k = 8* |
|---|---|---|---|---|---|---|---|
| **40** | **30** | **20** | **10** | **-40** | **-30** | **-20** | **-10** |

*Figura 4.3* Vector $V$ para intercambio

Luego, al intercambiar los elementos el primero con el último, y así sucesivamente, se debe utilizar un espacio temporal de intercambio para obtener un vector con contenido de informaciones mostrados en la figura 4.4.

| *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |
|---|---|---|---|---|---|---|---|
| **-10** | **-20** | **-30** | **-40** | **10** | **20** | **30** | **40** |

*Figura 4.4* Resultado del vector $V$ intercambiado

**Proc**: Intercambiador de elementos del vector $V$ de $k$ elementos
**Entero:** $k$, $i$, $V[50]$, $j$, $m$, $temp$, $n$
**Lea** $k$ // Leer el número de elementos del vector $V$
// Ciclo repetitivo que lee los elementos del vector
**Para**$(i = 1, k, 1)$**Haga**
  **Lea** $V[i]$ // Leer los elementos del vector identificado como $V$
**Fin_Para**
$j \leftarrow k$ // Subíndice $j$ que controla los elementos de la derecha
$m \leftarrow k/2$ // Punto medio del vector $V$
// Ciclo repetitivo que hace el intercambio de los elementos del vector
**Para**$(i = 1, m, 1)$**Haga**
  // $i$ es el subíndice que controla los elementos de la izquierda
  $temp \leftarrow V[i]$
  $V[i] \leftarrow V[j]$
  $V[j] \leftarrow temp$
  $j \leftarrow j - 1$
**Fin_Para**
1

1
// Ciclo repetitivo que imprime los elementos del vector $V$ intercambiados
**Para**($n = 1, k, 1$)**Haga**
// Imprime los elementos intercambiados del vector $V$
**Escriba** $V[n]$
**Fin_Para**
**Fin_proc**

---

Ejemplo 4.3 Cálculo del promedio acumulado semestral

---

Un estudiante matricula un conjunto $n$ de asignaturas. Para cada una de las asignaturas se conoce: el código de la asignatura, la calificación de la asignatura y el número de créditos asociados a la asignatura. Suponiendo que el número de asignaturas cursadas semestralmente por el alumno es menor o igual a cinco (5) en su ciclo de pregrado, diseñe las estructuras de datos en vectores columna donde se muestren la totalidad de los datos de entrada y el cálculo parcial de cada una de las contribuciones de las asignaturas al promedio, y construya un algoritmo para calcular el promedio semestral del estudiante.

| Subíndice | Código | Calificación | | Créditos | | Parcial |
|---|---|---|---|---|---|---|
| 1 | 1101 | 3,0 | * | 5 | = | 15 |
| 2 | 0010 | 5,0 | * | 1 | = | 5 |
| 3 | 2088 | 4,0 | * | 3 | = | 12 |
| 4 | 1120 | 3,0 | * | 3 | = | 9 |
| 5 | 3020 | 5,0 | * | 3 | = | 15 |
| Sumas | | | | 15 | | 56 |

*Figura 4.5* Vectores columna bases para el cálculo del promedio

*Análisis*: El promedio semestral de un estudiante (figura 4.5) se calcula multiplicando las calificaciones de las asignaturas cursadas durante el semestre por sus créditos asociados, y estos resultados se suman y se dividen por el total de créditos cursados por el alumno. El diseño de las estructuras de datos son cuatro vectores columna, con un rango de subíndice que varía de 1 hasta 5, por las cinco asignaturas que se disponen por filas en las estructuras.

Luego el promedio del alumno es igual a $56/15 = 3.73$.

**Proc**: Cálculo del promedio semestral
**Entero:** $n$, $i$, $Codigo[50]$, $Credito[50]$, $sumc$
**Real:** $Promedio$, $Parcial[50]$, $Calif[50]$, $suma$
**Lea** $n$
// Ciclo repetitivo que lee los códigos, calificaciones
1

```
1
// y créditos por asignatura
Para(i = 1, n, 1)Haga
    Lea Codigo[i], Calif[i], Credito[i]
Fin_Para
// Acumula los valores parciales de las calificaciones
// por los créditos
suma ← 0
sumc ← 0  // Variable que acumula la suma de los créditos
// Ciclo repetitivo que calcula los valores parciales de
// cada asignatura
Para(i = 1, n, 1)Haga
    Parcial[i] ← Calif[i] * Credito[i]
    suma ← suma + Parcial[i]
    sumac ← sumac + Credito[i]
Fin_Para
Promedio ← suma/sumac
// Ciclo repetitivo que imprime la justificación de los
// resultados parciales del promedio
Para(i = 1, n, 1)Haga
    Escriba Codigo[i], Calif[i], Credito[i], Parcial[i]
Fin_Para
Escriba "El valor del promedio acumulado del estudiante es "
Escriba Promedio
Fin_proc
```

Ejemplo 4.4 Búsqueda de un elemento $x$ en un vector leído de $n$ elementos

Dado un vector $V$ de $n$ elementos que contiene códigos representados por números de 4 dígitos, y dada una llave $X$, la cual es leída, diseñe un algoritmo para encontrar la llave $X$ en el vector $V$ utilizando la búsqueda secuencial identificando el subíndice en el que está ubicada la llave.

*Análisis*: Suponga $n = 10$ el número de elementos del vector $V$ y $X = 9071$, como se muestra en la figura 4.6. Entonces dentro del vector se debe variar el subíndice desde la posición inicial hasta la posición final con incrementos de 1, comparando el contenido de información con la llave $X$; y si el contenido de información coincide con la llave $X$, entonces se debe guardar el subíndice, punto en el cual se encuentra la llave buscada.

| *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
|---|---|---|---|---|---|---|---|---|---|
| **1922** | **3224** | **5456** | **5787** | **6908** | **7780** | **9071** | **5587** | **34** | **8999** |

*Figura 4.6* Búsqueda de un elemento $X$ en un vector

**Proc**: Búsqueda secuencial de una llave $X$ dentro de un vector $V$ de $n$ elementos

```
// Se leen el número de elementos del vector y la llave X
Entero: n, X, i, j, V[50], Swe, sub
Lea n, X
// Ciclo repetitivo que lee los elementos del vector V
Para(i = 1, n, 1)Haga
  Lea V[i]
Fin_Para
Swe ← 0 // Identifica cuando se encuentre la llave X
// Si la llave X se encuentra dentro del vector, el valor
// del Swe cambia su valor a 1
sub ← 0 // Subíndice del vector V en el que se encuentra el elemento X
// Ciclo repetitivo para que hace la búsqueda secuencial sobre el vector V
j ← 0
Swe ← 0
Mq(j < n y Swe = 0)Haga
  j ← j + 1
  Si(V[j] = X)Entonces
    Swe ← 1
    sub ← j
  F.Si
Fin_Mq
Si(Swe = 1)Entonces
  Escriba "El elemento ",X, " se encuentra en la posición"
  Escriba sub, " del vector"
Sino
  Escriba "El elemento ",X, "no se encuentra en el vector"
F.Si
Fin_proc
```

---

Ejemplo 4.5 Ordenamiento de un vector $T$ de $m$ elementos

Sea $m$ el número de elementos tipo real (con decimales) de un vector $T$, el cual debe ser leído, construya un algoritmo para ordenar e imprimir los elementos del vector $T$.

*Análisis*: Sea $m = 5$ el número de elementos del vector $T$ según se presenta en la figura 4.7, el método de ordenamiento que se va a utilizar es seleccionar la posición o el índice menor del vector que contiene la posición de la menor información del vector, en un ciclo interno, el cual es controlado por otro ciclo externo. El ciclo externo recorre el vector desde la primera posición hasta la última posición del vector, y el subíndice utilizado es $i$. Posteriormente el ciclo interno que detecta el índice del elemento menor del vector, ciclo que es controlado por $j$, que va desde el valor $i$ hasta el valor $m$

del vector. Al término de ciclo interno se tiene el índice menor, y la información contenida en este se pasa a un área temporal identificada como *temp*, que tiene como objetivo hacer el intercambio entre la información contenida en el índice menor con la información contenida en la posición $i$ del ciclo externo, y así sucesivamente hasta obtener el ordenamiento.

| *1* | *2* | *3* | *4* | *m=5* |
|---|---|---|---|---|
| **8.4** | **2.7** | **3.4** | **1.0** | **3.0** |

*Figura 4.7* Imagen inicial del ordenamiento de un vector $T$

Indice Menor = 4 $\implies$ $temp \leftarrow T[4]$; $T[4] \leftarrow T[i = 1]$ y finalmente a $T[i = 1] \leftarrow temp$. En la primera pasada del ciclo Para externo con $i = 1$ se selecciona el índice menor en el ciclo Para interno, o sea, el índice 4 que contiene la información 1; luego 1.0 se pasa a un área temporal, con el fin de intercambiarlo con la información contenida en la posición $i$ ($i = 1$, o sea, 8.4), y así sucesivamente hasta que se cumple el flujo de control de los dos ciclos anidados; al final se obtiene el vector ordenado como se presenta en la figura 4.8.

| *1* | *2* | *3* | *4* | *m=5* |
|---|---|---|---|---|
| **1.0** | **2.7** | **3.0** | **3.4** | **8.4** |

*Figura 4.8* Vector $T$ ordenado

**Proc**: Ordenamiento por selección del menor índice
| **Entero:** $m$, $i$, $im$, $j$
| **Real:** $T[50]$, $temp$
| **Lea** $m$ // Número de elementos del vector $T$
| // Ciclo repetitivo que lee los elementos del vector $T$
| $i \leftarrow 1$
| **Mq**$(i \leq m)$**Haga**
| | **Lea** $T[i]$
| | $i \leftarrow i + 1$
| **Fin_Mq**
| // Ciclo repetitivo externo que recorre el vector con subíndice $i$
| **Para**$(i = 1, m, 1)$**Haga**
| | // Subíndice que selecciona la posición del menor elemento
| | $im \leftarrow i$
| | **Para**$(j = i + 1, m, 1)$**Haga**
| | |
1 2 3

```
1 2 3
| | | Si(T[j] < T[im])Entonces
| | | | im ← j
| | | F.Si
| | Fin_Para
| | temp ← T[im]
| | T[im] ← T[i]
| | T[i] ← temp
| Fin_Para
| // Ciclo repetitivo que imprime el vector T ordenado
| Para(i = 1, m, 1)Haga
| | Escriba T[i]
| Fin_Para
Fin_proc
```

En la teoría de algoritmos existen otros métodos de ordenamiento que sirven para ordenar vectores con mayor o menor rapidez en los tiempos del proceso del ordenamiento, tales como los algoritmos de *Shellsort*, el ordenamiento por base, el ordenamiento de burbuja (*Bubble Sort*) y el ordenamiento rápido (*Quicksort*), entre otros, cuyas diferencias con el algoritmo expuesto radican en la forma como seleccionan los elementeos del vector en su proceso de ordenamiento. El ordenamiento de burbuja opera

> ... efecutando varias pasadas por el arreglo, comparando pares de clave en posiciones adyacentes e intercambiando sus elementos si no están en el orden. (Baase & Van, 2002, pp. 197-222).

Esta referencia es dada en virtud de la importancia del tópico de ordenamiento correpondiente a la Algoritmia.

Ejemplo 4.6 Operaciones elementos de vectores unidimensionales

Dados dos vectores ordenados ascendentemente, $A$ y $B$, de $n$ y $m$ elementos, respectivamente, los cuales contienen informaciones de número enteros no repetidos, suponiendo que $n \geq m$, diseñe un algoritmo para realizar las siguientes operaciones: $i$) contar el número de elementos del vector $A$ que aparecen en el vector $B$; $ii$) contar el número de elementos del vector $B$ que son diferentes de los del vector $A$; $iii$) crear un nuevo vector $C$ con los elementos que están en el vector $A$ pero que no pertenecen al vector $B$; $iv$) construir un nuevo vector $D$ con los elementos que aparecen en el vector $B$ pero no aparecen en el vector $A$.

*Análisis*: Dados el vector $A$ de $n$ igual a 8 elementos y el vector $B$ de 5 elementos, mostrados en la figura 4.9.

Luego, el número de elementos del vector $A$ que pertenecen al vector $B$ son tres (_): el 20, el 35 y el 77.

$A$:

| *1* | *2* | *3* | *4* | *5* | *6* | *7* | *n=8* |
|---|---|---|---|---|---|---|---|
| **20** | **30 (**)** | **35** | **50 (**)** | **60 (**)** | **68 (**)** | **77** | **89 (**)** |

$B$:

| *1* | *2* | *3* | *4* | *m=5* |
|---|---|---|---|---|
| **10 (*)** | **20** | **35** | **77** | **92 (*)** |

*Figura 4.9* Operaciones con vectores unidimensionales

El número de elementos del vector $B$ que son diferentes del vector $A$ son dos (*): el número 10 y el 92.

Los elementos del vector $A$ que no pertenecen a $B$ son cinco (**) y están contenidos en $C$

$C$:

| *1* | *2* | *3* | *4* | *5* |
|---|---|---|---|---|
| **30** | **50** | **60** | **68** | **89** |

Finalmente, los elementos del vector $B$ que no aparecen en el vector $A$ son dos (*) y que sirven para formar el vector $D$ son

$D$:

| *1* | *2* |
|---|---|
| **10** | **92** |

**Proc**: Operaciones con vectores unidimensionales
| **Entero:** $n$, $m$, $i$, $j$, $A[50]$, $B[50]$, $conab$, $conba$, $k$, $p$, $swe$, $swd$, $C[50]$, $D[50]$
| **Lea** $n, m$ // Número de elementos de los vectores $A$ y $B$
| // Lectura del vector $A$
| **Para**$(i = 1, n, 1)$**Haga**
| | **Lea** $A[i]$
| **Fin_Para**
| // Lectura del vector $B$
| **Para**$(i = 1, m, 1)$**Haga**
| | **Lea** $B[i]$
| **Fin_Para**
| $conab \leftarrow 0$ // Cuenta los elementos de $A$ en $B$
| $conba \leftarrow 0$ // Cuenta los elementos de $B$ que no están en $A$
| $k \leftarrow 0$ // Controla la construcción del vector $C$
| $p \leftarrow 0$ // Controla la construcción del vector $D$
| $i \leftarrow 1, j \leftarrow 1$
| **Mq**$(i \leq n)$**Haga**
| | $swe \leftarrow 0$
1 2

```
1 2
    Mq(j ≤ m)Haga
      Si(A[i] = B[j])Entonces
        conab ← conab + 1
        swe ← 1
      F.Si
      j ← j + 1
    Fin_Mq
    Si(swe ≠ 1)Entonces
      k ← k + 1
      C[k] ← A[i]
    F.Si
    i ← i + 1, j ← 1
  Fin_Mq
  j ← 1, i ← 1
  Mq(j ≤ m)Haga
    swd ← 1
    Mq(i ≤ n y swd = 1)Haga
      Si(B[j] = A[i])Entonces
        swd ← 0
      F.Si
      i ← i + 1
    Fin_Mq
    Si(swd = 1)Entonces
      conba ← conba + 1
      p ← p + 1
      D[p] ← B[j]
    F.Si
    j ← j + 1, i ← 1
  Fin_Mq
  Escriba "El número de elementos de A que están en B es ", conab
  Escriba "El número de elementos de B que no están en A es ", conba
  Escriba "Los elementos del vector A que aparecen en el vector B son "
  Para(i = 1, k, 1)Haga
    Escriba C[i]
  Fin_Para
  Escriba "Los elementos del vector B que no están en el vector A son "
  Para(j = 1, p, 1)Haga
    Escriba D[j]
  Fin_Para
Fin_proc
```

Ejemplo 4.7 Conversión de un número de base 10 a base 2 sobre un vector Resultado

Dado un número $n$ en base 10, el cual debe ser leído, construya un algoritmo para convertir el número dado de base 10 a base 2, dejando el resultado en un vector $R$.

*Análisis*: Sea $n = (7)_{10}$ denotado en base 10; entonces, como la base a la cual se va a convertir el número es 2, dividimos sucesivamente el número hasta que obtengamos un cociente de 1, y los residuos de cada división son los equivalentes del número decimal en binario. Explicitado de la siguiente forma: $7/2 = 3$, y el primer residuo es 1; $3/2 = 1$, y el segundo residuo es 1. Luego el número $(7)_{10}$ en base 10 es igual al cuarteto binario $(0111)_2$. Se debe tener en cuenta que los valores resultados de la base 2 se deben tomar en un sentido inverso, de la siguiente manera: sea $n = (8)_{10}$ al dar su equivalente en base 2, entonces $8/2 = 4$, y el residuo es 0 (primer residuo); $4/2 = 2$, y el residuo es 0 (segundo residuo); $2/2 = 1$, y el residuo (tercer residuo) es 0; entonces el resultado en base 2 es 1 y los residuos tomados en un sentido inverso, lo que es equivalente al cuarteto $(1000)_2$.

```
Proc: Conversor de base 10 a base 2
  Entero: n, n1, j, R[50], i
  Lea n
  n1 ← n  // Se define una variable temporal de n para conservarlo
  j ← 0  // Subíndice que va a guardar el resultado del residuo
  // en el vector R
  // Ciclo repetitivo de división el número entre dos
  Mq(n1/2 ≠ 1)Haga
    j ← j + 1
    R[j] ← n1 mod 2
    n1 ← n1/2
  Fin_Mq
  Si(n1 = 3)Entonces
    j ← j + 1
    R[j] ← 1
    j ← j + 1
    R[j] ← 1
  Sino
    j ← j + 1
    R[j] ← 0
    j ← j + 1
    R[j] ← 1
  F.Si
1
```

1

// Ciclo repetitivo que imprime el vector resultado $R$
**Para**($i = j, 1, -1$)**Haga**
  **Escriba** $R[j]$
**Fin_Para**
**Fin_proc**

Ejemplo 4.8 Cálculo de la media armónica (MH)

Suponga un conjunto de $n$ números almacenados en el vector $X$, a los cuales se les asocia sus respectivas frecuencias absolutas almacenados en el vector $F$, diseñe un algoritmo que calcule la media armónica del conjunto de $n$ números, leyendo los vectores $X$ y $F$, suponiendo que en los datos de entrada no existen valores nulos.

*Análisis*: La media armónica identificada por H (Tremblay & Daoust, 1989, p. 179) de una cantidad mensurable de números es igual al inverso de la media aritmética de los recíprocos de dichos valores, y se representa por la ecuación matemática

$$H = \frac{n}{\displaystyle\sum_{i=1}^{n} \frac{F_i}{X_i}}$$

Existe una desigualdad que relaciona la media aritmética, la media geométrica y la media armónica. Muchas veces se menciona como "MA-MG-MH". La desigualdad establece que dados $a_1, a_2, \ldots, a_n$ números reales positivos, se tiene que

$$\frac{a_1 + a_2 + \cdots + a_n}{n} \geq \sqrt[n]{a_1 a_2 \cdots a_n} \geq \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \cdots + \frac{1}{a_n}}$$

**Proc**: Cálculo de la media armónica
**Entero:** $n$, $i$
**Real:** $X[50]$, $F[50]$, $d$, $H$
**Lea** $n$
**Para**($i = 1, n, 1$)**Haga**
  **Lea** $X[i], F[i]$
**Fin_Para**
// Ciclo repetitivo que calcula el denominador de la media armónica
$d \leftarrow 0$
**Para**($i = 1, n, 1$)**Haga**
  $d \leftarrow d + F[i]/X[i]$
**Fin_Para**

1

1
$H \leftarrow n/d$
**Escriba** "El valor de la media armónica es igual a ", $H$
**Fin_proc**

---

Ejemplo 4.9 Generar los $k$ primeros números primos en un vector de $k$ elementos

Dado un vector $A$ de $k$ elementos, *llene* el vector con los $k$ primeros números primos.

*Análisis*: Supuesto $k = 5$ elementos, un número primo es aquel que únicamente es divisible por sí mismo y por la unidad, como es el caso de los números del vector $A$ en la figura 4.10.

$A$:

| *1* | *2* | *3* | *4* | *5* |
|---|---|---|---|---|
| **2** | **3** | **5** | **7** | **11** |

*Figura 4.10* Generación de $k$ primeros números primos

**Proc**: Generación de los $k$ primeros números primos
  **Entero:** $k$, $i$, $A[50]$, $num$, $swh$, $swp$, $j$
  **Lea** $k$
  $num \leftarrow 1$
  $A[1] \leftarrow 2$
  // Ciclo repetitivo generador de los números primos
  $i \leftarrow 2$
  **Mq**($i \leq k$)**Haga**
    $swh \leftarrow 0$
    **Mq**($swh = 0$)**Haga**
      $j \leftarrow 2$
      $swp \leftarrow 0$
      $num \leftarrow num + 2$
      **Mq**($j \leq \sqrt{num}$ y $swp = 0$)**Haga**
        **Si**($num$ **mod** $j = 0$)**Entonces**
          $swp \leftarrow 1$
        **F.Si**
        $j \leftarrow j + 1$
      **Fin_Mq**
      **Si**($swp = 0$)**Entonces**
        $A[i] \leftarrow num$
        $swh \leftarrow 1$
      **F.Si**
    **Fin_Mq**
1 2

```
1 2
| | i ← i + 1
| Fin_Mq
| // Ciclo repetitivo impresor de los k primeros números primos
| Para(j = 1, k, 1)Haga
| | Escriba A[j]
| Fin_Para
Fin_proc
```

---

Ejemplo 4.10 Mezcla de vectores ordenados

---

Se tienen dos vectores, $A$ y $B$, cada uno de $n$ y $m$ elementos, respectivamente, los cuales se encuentran ordenados ascendentemente y contienen números enteros positivos. Diseñe un algoritmo para mezclar los dos vectores en un nuevo vector $C$ de $n+m$ elementos tal que el vector $C$ quede ordenado ascendentemente.

*Análisis*: Suponga $A$ un vector unidimensional de 5 elementos ($n = 5$) y $B$ un vector de 8 elementos ($m = 8$), como se muestran en la figura 4.11.

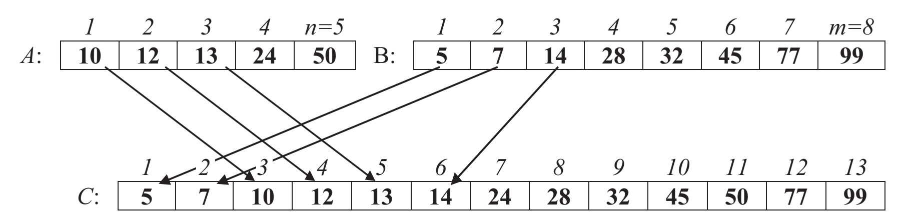



*Figura 4.11* Vectores ordenados ascendentemente

Los dos vectores están ordenados ascendentemente, en tal sentido es la primera verificación que debe hacer el algoritmo; si los vectores no están ordenados, entonces no se puede realizar su proceso de mezcla. Para realizar la mezcla se comparan el primer elemento del vector $A$ con el primer elemento del vector $B$ y se selecciona el elemento menor. En este caso $A[1] = 10$ y $B[1] = 5$; y se asigna el menor elemento al vector $C$ en su primera posición. Luego $C[1] = 5$; en caso contrario se asigna al vector $C$ el elemento del vector $A$, y se continúa el proceso hasta que se alcancen los topes de cualquiera de los dos vectores, lo cual se controla con el subíndice ($i$) que va llenando los elementos resultados de la mezcla en el vector $C$.

Posteriormente, con base en el subíndice ($i$), se completan en el vector $C$ los elementos faltantes del vector que no terminó la mezcla, ubicándolos en posiciones posteriores al valor del subíndice $i$ hasta que se alcance el valor de $n$ o de $m$ que hicieron que el valor de $i$ haya llegado a su tope en el proceso anterior.

```
Proc: Mezcla de dos vectores ordenados, A y B, en C
  Entero: n, m, i, j, k, p, A[50], B[50], C[50], SwOA, SwOB
  Lea n, m
  Para(i = 1, n, 1)Haga
    Lea A[i]
  Fin_Para
  Para(i = 1, m, 1)Haga
    Lea B[i]
  Fin_Para
  // Se comprueba si el vector A está ordenado
  SwOA ← 0
  i ← 1
  Mq(i < n y SwOA = 0)Haga
    Si(A[i] > A[i + 1])Entonces
      SwOA ← 1
    F.Si
    i ← i + 1
  Fin_Mq
  // Se comprueba si el vector B está ordenado
  SwOB ← 0
  i ← 1
  Mq(i < m y SwOB = 0)Haga
    Si(B[i] > B[i + 1])Entonces
      SwOB ← 1
    F.Si
    i ← i + 1
  Fin_Mq
  // Chequeo que la mezcla se puede hacer cuando los dos vectores
  // están ordenados
  Si(SwOA = 0 y SwOB = 0)Entonces
    // j, k, i son índices para los vectores A, B, C, respectivamente
    j ← 1, k ← 1, i ← 1
    // Determinar el menor de los elementos entre A y B
    // y almacenar en C, de tal forma que la cuando se unan
    // los vectores, el resultado también esté ordenado
    Mq(j ≤ n y k ≤ m)Haga
      Si(A[j] ≤ B[k])Entonces
        // El menor elemento se encuentra en A
        C[i] ← A[j]
        j ← j + 1
        i ← i + 1
      Sino
        // El menor elemento se encuentra en B
1 2 3 4
```

```
1 2 3 4
│ │ │ │ C[i] ← B[k]
│ │ │ │ k ← k + 1
│ │ │ │ i ← i + 1
│ │ │ F.Si
│ │ Fin_Mq
│ │ // Realiza la copia de los elementos faltantes
│ │ Si(j > n)Entonces
│ │ │ // Todos los elementos de A fueron mezclados
│ │ │ Para(p = k, m, 1)Haga
│ │ │ │ C[i] ← B[p]
│ │ │ │ i ← i + 1
│ │ │ Fin_Para
│ │ Sino
│ │ │ // Todos los elementos de B fueron mezclados
│ │ │ Para(p = j, n, 1)Haga
│ │ │ │ C[i] ← A[p]
│ │ │ │ i ← i + 1
│ │ │ Fin_Para
│ │ F.Si
│ │ // Escritura de los vectores originales y su mezcla
│ │ Para(i = 1, n, 1)Haga
│ │ │ Escriba A[i]
│ │ Fin_Para
│ │ Para(i = 1, m, 1)Haga
│ │ │ Escriba B[i]
│ │ Fin_Para
│ │ Para(i = 1, n + m, 1)Haga
│ │ │ Escriba C[i]
│ │ Fin_Para
│ Sino
│ │ Escriba "Los vectores A y B no están ordenados"
│ F.Si
Fin_proc
```

Ejemplo 4.11 Búsqueda de patrones

Suponga un vector $V$ de $n$ elementos identificado como arreglo unidimensional base y otro vector identificado como $P$ de $m$ elementos, llamado "patrón". Diseñe un algoritmo que cuente el número de veces que los elementos del patrón aparecen en el vector base $V$.

*Análisis*: Sea $V$ un vector de $n = 7$ elementos, y $P$ un vector de $m = 2$ elementos, tal y como se muestran en la figura 4.12.

Luego, el número de veces que aparece el patrón $P$ en el vector $V$ son dos veces (como se muestra, a nivel subrayado, en el vector base $V$), porque $P[1] = V[1]$ y $P[2] = V[2]$; pero adicionalmente $P[1] = V[6]$ y $P[2] = V[7]$.

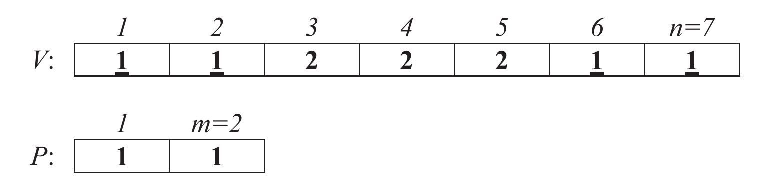



*Figura 4.12* Búsqueda de patrones de un vector $P$ en un vector $V$

```
Proc: Búsqueda de patrones de un vector P en un vector V
| Entero: n, m, i, j, V[50], P[50], contp, swp
| Lea n, m
| Para(i = 1, n, 1)Haga
| | Lea V[i]
| Fin_Para
| Para(i = 1, m, 1)Haga
| | Lea P[i]
| Fin_Para
| contp ← 0 // Acumula el número de veces que P está en V
| i ← 1
| Mq(i ≤ n)Haga
| | swp ← 0
| | j ← 1
| | Mq(j ≤ m y swp = 0)Haga
| | | Si(V[i] = P[j])Entonces
| | | | i ← i + 1
| | | | j ← j + 1
| | | Sino
| | | | swp ← 1
| | | F.Si
| | Fin_Mq
| | Si(swp = 0 y j > m)Entonces
| | | contp ← contp + 1
| | F.Si
| | i ← i + 1
| Fin_Mq
| // Escritura del vector base V
| Para(i = 1, n, 1)Haga
| | Escriba V[i]
| Fin_Para
| // Escritura del vector de patrones
1
```

```
1
| Para(i = 1, m, 1)Haga
| | Escriba P[i]
| Fin_Para
| // Escritura del resultado del conteo de patrones
| Escriba “Los patrones contenidos en el vector V son ”, contp
Fin_proc
```

# 4.2 Concepto de arreglo bidimensional o matriz

El concepto de *arreglo unidimensional* permite almacenar un elemento de información en un vector; elemento que está contenido en la dirección identificada por el subíndice del vector, que necesariamente es un solo un subíndice lineal. Sea $A$ un vector unidimensional de $n$ elementos que contiene números binarios (1s y 0s) direccionados por el subíndice $i$. Entonces $A[i]$ para $i$ desde 1 hasta el valor $n$ contiene las informaciones binarias almacenadas en el vector. Los vectores de una sola dimensión (o arreglos unidimensionales) son la base para el concepto de arreglo bidimensional.

Un *arreglo bidimensional* es un conjunto de elementos de la misma especie contenidos en un conjunto de vectores filas o líneas, los cuales se intersectan con otro conjunto de vectores columna; y el espacio de intersecto fila $\cap$ columna genera un lugar para el almacenamiento de la información; espacio que es direccionado por dos subíndices que identifican la posición dentro del arreglo en el que se encuentra el elemento de información.

Las características de los vectores fila y columna, que se intersectan para conformar un arreglo bidimensional, tienen las mismas características que fueron enunciadas para los vectores unidimensionales, en el sentido de que los dos subíndices del arreglo bidimensional tienen una estructura lineal de almacenamiento de las informaciones contenidas en el arreglo.

El arreglo bidimensional está compuesto por los siguientes elementos:

- La identidad del arreglo, o nombre del arreglo, que identifica la estructura de datos llamada matriz, arreglo bidimensional, o arreglo en dos dimensiones.
- Un primer subíndice, que direcciona los vectores filas o líneas.
- Un segundo subíndice, que direcciona los vectores columnas.
- La intersección del vector fila con el vector columna, la cual es la relación cruzada fila y columna que genera el espacio de almacenamiento y permite almacenar el elemento de información.

- Las informaciones contenidas en el arreglo bidimensional, las cuales son direccionadas tanto por la identidad del arreglo, como por las direcciones del cruces de las filas con las columnas.

- La dimensión del arreglo o espacio de almacenamiento compuesto por la relación cruzada del número de vectores fila y el número de vectores columna, lo cual puede generar arreglos bidimensionales cuadrados cuando el número de filas es igual al número de columnas.

Con base en lo anterior, un elemento de un arreglo bidimensional o en dos dimensiones "... puede ser accedido mediante la especificación de dos índices el número de la fila y el número de la columna" (Tenenbaum & Augestein, 1981, p. 27).

Sea $B$ un arreglo bidimensional compuesto por la intersección de tres vectores línea con tres vectores columna. Sea $i$ el subíndice de direccionamiento de los vectores fila y $j$ el subíndice de direccionamiento de los vectores columna. Luego el rango de los subíndices mencionados está contenido en el intervalo cerrado $1 \leq i, j \leq 3$. Matemáticamente, los elementos $b_{ij}$ que componen el arreglo bidimensional $B$ de nueve elementos se disponen según la figura 4.13.

| $B_{nn=3\times 3}$ | Columna $j=1$ | Columna $j=2$ | Columna $j=3$ |
|---|---|---|---|
| Fila $i=1$ | $b_{11}$ | $b_{12}$ | $b_{13}$ |
| Fila $i=2$ | $b_{21}$ | $b_{22}$ | $b_{23}$ |
| Fila $i=3$ | $b_{31}$ | $b_{32}$ | $b_{33}$ |

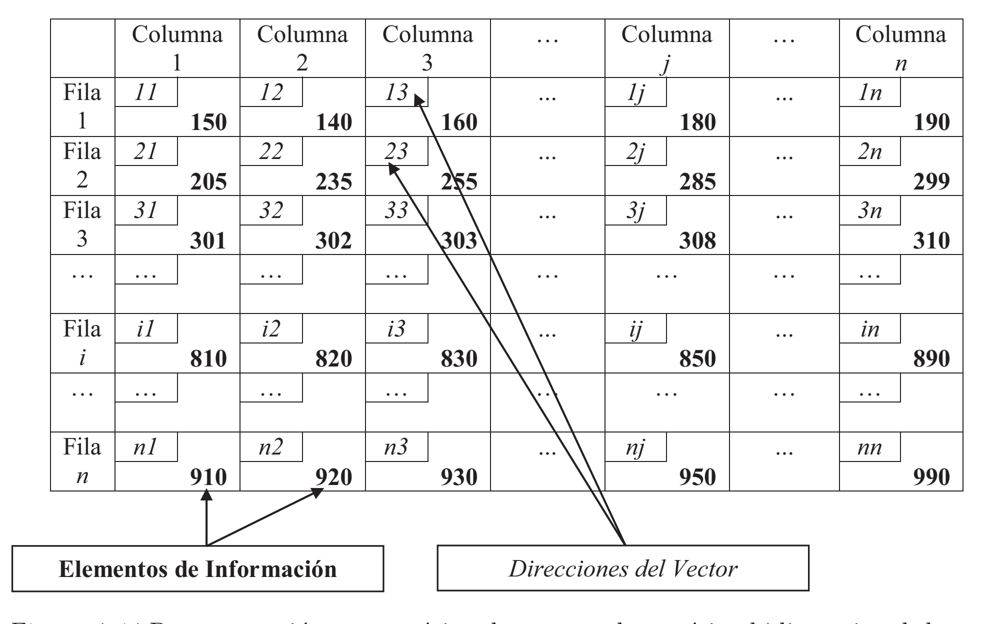
*Figura 4.13* Representación matemática de los elementos que componen el arreglo bidimensional $B$.

Esquemáticamente, el arreglo bidimensional o matriz de $n$ filas por $n$ columnas se representa en la figura 4.14.

Con base en la figura 4.14, el arreglo bidimensional está identificado por el nombre $V$. El arreglo o matriz tiene $n \times n$ elementos. Los elementos de información están direccionados en sus filas por el subíndice $i$ y en sus columnas por el subíndice $j$; de tal manera que en la dirección de la fila dos y columna 3 se encuentra la información 255; es decir, $V[2, 3] = 255$.

Las estructuras lógicas que se utilizan para manejar los arreglos bidimensionales son las mismas que se han utilizado en las secciones anteriores de este texto. Por consiguiente, sobre una matriz se puede: leer valores para asignarlos a una posición $(i, j)$ de la matriz, asignar valores a una posición del arreglo bidimensional, escribir los valores de una o varias posiciones de la matriz; y recorrer la matriz utilizando las direcciones de sus posiciones, lo cual se puede realizar con las estructuras lógicas de control: **Para, Mientras que** o **Haga Hasta**.

$V(i,j)$:



| | Columna 1 | Columna 2 | Columna 3 | … | Columna *j* | … | Columna *n* |
|---|---|---|---|---|---|---|---|
| Fila 1 | *11* **150** | *12* **140** | *13* **160** | ... | *1j* **180** | ... | *1n* **190** |
| Fila 2 | *21* **205** | *22* **235** | *23* **255** | ... | *2j* **285** | ... | *2n* **299** |
| Fila 3 | *31* **301** | *32* **302** | *33* **303** | ... | *3j* **308** | ... | *3n* **310** |
| … | … | … | … | … | … | … | … |
| Fila *i* | *i1* **810** | *i2* **820** | *i3* **830** | ... | *ij* **850** | ... | *in* **890** |
| … | … | … | … | … | … | … | … |
| Fila *n* | *n1* **910** | *n2* **920** | *n3* **930** | ... | *nj* **950** | ... | *nn* **990** |



*Figura 4.14* Representación esquemática de un arreglo genérico bidimensional de $n$ filas por $n$ columnas con sus elementos de información asociadas con la identidad del vector $V$

Ejemplo 4.12 Lectura y escritura de los elementos de un arreglo bidimensional

Dado un arreglo bidimensional de $k \times k$ elementos llamado $A$, el cual almacena informaciones de números enteros, diseñe un algoritmo que lea los elementos del arreglo por columnas e imprima los elementos de la matriz por filas.

*Análisis*: Suponga $k = 3$, entonces el arreglo bidimensional tendrá 9 elementos, los cuales son leídos por columnas, supuestos los elementos de la matriz iguales a 10, 20, 30, 40, 50, 60, 70 80 y 90. Entonces los elementos deben quedar almacenados en el arreglo así: los tres primeros en la primera columna, los tres segundos en la segunda columna y así sucesivamente, según la figura 4.15.

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 10 | 40 | 70 |
| 2 | 20 | 50 | 80 |
| 3 | 30 | 60 | 90 |

*Figura 4.15* Almacenamiento de los elementos del arreglo $A$ por columnas

Por su parte, la escritura de los elementos del arreglo se deben imprimir por filas, luego varía más rápido el subíndice que controla las columnas (→).

**Proc**: Almacenamiento de los elementos de un arreglo $A$ por columnas y escritura por filas

```
  Entero: k, i, j, A[50, 50]
  Lea k
  // Ciclo repetitivo que lee los elementos por columnas
  Para(j = 1, k, 1)Haga
    Para(i = 1, k, 1)Haga
      // Varían más rápido las filas i
      Lea A[i, j]
    Fin_Para
  Fin_Para
  // Ciclo repetitivo para que escribe los elementos por filas
  Para(i = 1, k, 1)Haga
    Para(j = 1, k, 1)Haga
      // Varían más rápido las columnas
      Escriba A[i, j]
    Fin_Para
  Fin_Para
Fin_proc
```

Ejemplo 4.13 Generación de la matriz transpuesta

Sea una matriz $B$ de $n \times n$ elementos, la cual se debe leer, diseñe un algoritmo para hallar la transpuesta de la matriz.

*Análisis*: Si $B$ es la matriz de $n \times n$ a la cual se le va a hallar su transpuesta, entonces para hallar $B^t$, o matriz transpuesta, se intercambian las filas con las columnas; o sea, $b_{ij}$ en $B$ es igual a $b_{ji}$, según se presenta en la figura 4.16.

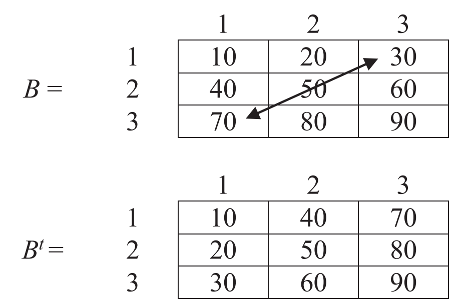



*Figura 4.16* Matriz transpuesta

```
Proc: Transpuesta de una matriz
  Entero: n, i, j, B[50, 50], temp
  Lea n
  // Ciclo repetitivo para que lee los elementos del arreglo B por filas
  Para(i = 1, n, 1)Haga
    Para(j = 1, n, 1)Haga
      Lea B[i, j]
    Fin_Para
  Fin_Para
  // Ciclos repetitivos que realizan la transpuesta de la matriz B
  Para(i = 1, n, 1)Haga
    Para(j = 1, i − 1, 1)Haga
      temp ← B[i, j]
      B[i, j] ← B[j, i]
      B[j, i] ← temp
    Fin_Para
  Fin_Para
  Para(i = 1, n, 1)Haga
    Para(j = 1, n, 1)Haga
      Escriba B[i, j]
    Fin_Para
  Fin_Para
Fin_proc
```

Ejemplo 4.14 Operador matricial

Dadas dos matrices $A$ y $B$ de $n \times n$ elementos, las cuales son leídas, diseñe un algoritmo para realizar las operaciones de $i$) Suma de matrices sobre la matriz $C$ ($C = A+B$). $ii$) Resta de matrices sobre la matriz $D$ ($D = A-B$). $iii$) Multiplicación de matrices sobre la matriz $E$ ($E = A * B$).

*Análisis*: Supuesto $n = 3$, entonces el proceso de cada una de las operaciones se presenta a continuación:

$i$) Los valores y el resultado de la suma de matrices se muestran a continuación. Sean

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Por lo tanto,

$$C = A + B = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Con base en lo anterior, cada elemento de la matriz $C$ es igual al elemento de la matriz $A$, sumado con el elemento de la matriz $B$ en la misma posición; o sea, $C(i,j) = A(i,j) + B(i,j)$, $1 \leq i,j \leq n$.

*ii*) Para el caso de la resta de matrices suponga

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Por lo tanto,

$$D = A - B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Luego el valor de $D$ es igual al elemento de $A$ menos el elemento de $B$, en la misma posición; por lo tanto, $D(i,j) = A(i,j) - B(i,j)$.

*iii*) La multiplicación de matrices con la dimensión de la matriz de $3 \times 3$ se presenta a continuación. Suponga,

$$A = B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Por lo tanto,

$$E = A * B = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

Con base en lo anterior, cada elemento de la matriz

$$E(i,j) = \sum_{k=1}^{n} A(i,k) * B(k,j), \quad \text{con } 1 \leq i,j \leq n.$$

**Proc**: Operador matricial de la suma, resta y multiplicación de dos matrices, $A$ y $B$

```
| Entero: n, i, j, A[50,50], B[50,50], C[50,50], D[50,50], E[50,50]
| Lea n
| Para(i = 1, n, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | Lea A[i,j], B[i,j]
| | Fin_Para
| Fin_Para
| Para(i = 1, n, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | // Cálculo de C = A + B
| | | C[i,j] ← A[i,j] + B[i,j]
1 2 3
```

```
1 2 3
|  |  |  // Cálculo de C = A − B
|  |  |  D[i, j] ← A[i, j] − B[i, j]
|  |  |  // Cálculo de C = A ∗ B
|  |  |  E[i, j] ← 0
|  |  |  Para(k = 1, n, 1)Haga
|  |  |  |  E[i, j] ← E[i, j] + A[i, k] ∗ B[k, j]
|  |  |  Fin_Para
|  |  Fin_Para
|  Fin_Para
|  // Impresión de resultados
|  Escriba "Suma de las matrices A y B"
|  Para(i = 1, n, 1)Haga
|  |  Para(j = 1, n, 1)Haga
|  |  |  Escriba C[i, j]
|  |  Fin_Para
|  Fin_Para
|  Escriba "Resta de las matrices A y B"
|  Para(i = 1, n, 1)Haga
|  |  Para(j = 1, n, 1)Haga
|  |  |  Escriba D[i, j]
|  |  Fin_Para
|  Fin_Para
|  Escriba "Producto de las matrices A y B"
|  Para(i = 1, n, 1)Haga
|  |  Para(j = 1, n, 1)Haga
|  |  |  Escriba E[i, j]
|  |  Fin_Para
|  Fin_Para
Fin_proc
```

Ejemplo 4.15 Chequeo de una matriz cuadrado mágico

Dada una matriz de $n \times n$, la cual debe ser leída, diseñe un algoritmo para chequear si el arreglo bidimensional dado es un cuadrado mágico. Una matriz es cuadrado mágico si la suma de sus filas, columnas, diagonal principal y la diagonal secundaria es igual al mismo número.

*Análisis*: Sea la matriz $A$ de $n \times n$ elementos para $n = 5$

$$A = \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}$$

Se nota en la matriz $A$ que los elementos de las cinco columnas suman 65, al igual que los valores ubicados en las cinco líneas; como también la suma de los elementos de la diagonal principal (DP) y de la diagonal secundaria (DS) da un valor igual a 65. Por lo tanto, la matriz es un cuadrado mágico.

```
Proc: Chequeo de matriz cuadrado mágico
  Entero: n, i, j, A[50, 50], DP, DS, swm
  Lea n
  Para(i = 1, n, 1)Haga
    Para(j = 1, n, 1)Haga
      Lea A[i, j]
    Fin_Para
  Fin_Para
  DP ← 0  // Suma de los elementos de la diagonal principal
  DS ← 0  // Suma los elementos de la diagonal secundaria
  // Para la suma de los elementos de las filas y las columnas
  // Se va a utilizar una matriz ampliada de (n + 1) × (n + 1)
  // En la matiz ampliada se colocan en ceros los elementos de la
  // columna n + 1 y los elementos de la fila n + 1 de la matriz A
  Para(i = 1, n, 1)Haga
    A[i, n + 1] ← 0
    A[n + 1, i] ← 0
  Fin_Para
  // Ciclos repetitivo que suma los elementos de filas, columnas
  // y diagonales
  Para(i = 1, n, 1)Haga
    Para(j = 1, n, 1)Haga
      A[i, n + 1] ← A[i, n + 1] + A[i, j]
      A[n + 1, j] ← A[n + 1, j] + A[i, j]
      Si(i = j)Entonces
        DP ← DP + A[i, j]
      F.Si
      Si(i + j = n + 1)Entonces
        DS ← DS + A[i, j]
      F.Si
    Fin_Para
  Fin_Para
  // Preguntas de chequeo de la matriz cuadrado mágico
  swm ← 0
  Para(i = 1, n, 1)Haga
    Si(A[i, n + 1] ≠ DP y A[i, n + 1] ≠ DS)Entonces
      swm ← 1
    F.Si
1 2
```

1 2

**Si**$(A[n+1,i] \neq DP$ y $A[n+1,i] \neq DS)$**Entonces**
$swm \leftarrow 1$
**F.Si**
**Si**$(DP \neq DS)$**Entonces**
$swm \leftarrow 1$
**F.Si**
**Fin_Para**
**Si**$(swm = 0)$**Entonces**
**Escriba** "La matriz dada es cuadrado mágico"
**Sino**
**Escriba** "La matriz dada no es cuadrado mágico"
**F.Si**
**Fin_proc**

---

Ejemplo 4.16 Matriz ordenada descendentemente

Lea una matriz con identidad $A$ de orden cuadrado que tiene $n$ elementos; diseñe un algoritmo para ordenar los elementos de la matriz en orden descendente de la primera $(A[1,1])$ a la última posición de la matriz $(A[n,n])$.

*Análisis*: Suponga la matriz $A$ de $n = 3$, como se presenta a continuación:

Matriz original

$$A = \begin{bmatrix} 3 & 2 & 5 \\ 4 & 1 & 6 \\ 8 & 7 & 9 \end{bmatrix}$$

Matriz orden descendente

$$A = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

El método de ordenamiento sobre la matriz que se va a utilizar es filtrar el primer mayor elemento de la matriz; en este caso, el número 9 ubicado en la última posición. Luego, el mayor se envía al frente de la tabla, en este caso, la posición $A[1,1]$ con información, la cual pasa a la posición en la cual se encontraba la información 9. Posteriormente se encuentra el segundo mayor y se intercambia con la posición en curso de la tabla después de ubicado el primer mayor, y así sucesivamente hasta que se hayan mayorizado los $n \times n$ elementos de la matriz.

Se hace la claridad para el método expuesto que hay lógicas de control de ordenamiento de matrices más eficientes; pero el texto está enfocado a presentar lógicas efectivas y no tiene como esencia el análisis de algoritmos en cuanto a los parámetros de evaluación de efectividad de las variables de tiempo y espacio de los algoritmos.

**Proc**: Ordenamiento de matriz descendente
**Entero:** $n$, $i$, $j$, $A[50]$, $temp$, $B[50,50]$
**Lea** $n$
1

```
1
| Para(i = 1, n, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | Lea A[n * (i − 1) + j]
| | Fin_Para
| Fin_Para
| // Ordenamiento del vector A
| Para(i = 1, n * n − 1, 1)Haga
| | Para(j = i + 1, n * n, 1)Haga
| | | Si(A[i] < A[j])Entonces
| | | | temp ← A[i]
| | | | A[i] ← A[j]
| | | | A[j] ← temp
| | | F.Si
| | Fin_Para
| Fin_Para
| // Creación de una matriz a partir del vector A
| Para(i = 1, n, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | B[i, j] ← A[n * (i − 1) + j]
| | Fin_Para
| Fin_Para
| // Impresión de la matriz
| Para(i = 1, n, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | Escriba B[i, j]
| | Fin_Para
| Fin_Para
Fin_proc
```

---

Ejemplo 4.17 Simulador del juego del baloto

Construya un algoritmo que simule $n$ sorteos del baloto en una matriz en la que cada sorteo corresponde a una línea de la matriz y las columnas de la matriz representan los valores de los números del baloto en el intervalo cerrado $[1, k = 45]$. Si se tiene en cuenta que se deben leer los $m$ números ($m = 6$) de cada sorteo, ordene descendentemente los números que más salieron en las $n$ simulaciones.

*Análisis*: Se van a simular 10 sorteos ($n = 10$), con números ($num$) en el intervalo cerrado de $1 \leq num \leq k = 10$, y cada sorteo tiene $m$ ($m = 6$) números ganadores (figura 4.17).

En el rango $[1, 10]$ el 5 salió el mayor número de veces y el 10 salió menos veces.

| Sorteos ↓ | Números ganadores | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $A(i,j)$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 1 | | | 1 | 1 | 1 | | 1 | 1 | |
| 2 | 1 | | | | 1 | 1 | 1 | 1 | 1 | |
| 3 | | 1 | 1 | 1 | | 1 | 1 | 1 | | |
| 4 | 1 | 1 | 1 | 1 | 1 | | | | | 1 |
| 5 | 1 | | 1 | | 1 | | 1 | | 1 | 1 |
| 6 | | 1 | 1 | | | 1 | 1 | 1 | | 1 |
| 7 | 1 | 1 | 1 | | 1 | 1 | 1 | | | |
| 8 | | | | 1 | 1 | 1 | | 1 | 1 | 1 |
| 9 | 1 | | 1 | | 1 | 1 | 1 | 1 | | |
| $n=10$ | 1 | 1 | 1 | 1 | 1 | | | | 1 | |

| Cont | 7 | 5 | 7 | 5 | 8 | 7 | 6 | 6 | 5 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|

*Figura 4.17* Tabla de simulación del juego del baloto

**Proc**: Simulador de baloto matricial
**Entero:** $n$, $k$, $m$, $num$, $i$, $j$, $A[50,50]$, $temp$, $Cont[50]$, $Num[50]$
**Lea** $n$ // Número de sorteos
**Lea** $k$ // Lee el límite del intervalo cerrado $[1, k = 45]$
**Lea** $m$ // Lee el número de números ganadores en cada sorteo ($m = 6$)
**Para**$(i = 1, n, 1)$**Haga**
  $j \leftarrow 0$
  **Mq**$(j \leq m)$**Haga**
    **Lea** $num$
    // Controlador de números válidos por sorteo
    **Si**$(num \geq 1$ y $num \leq k)$**Entonces**
      $A[i, num] \leftarrow 1$
      $j \leftarrow j + 1$
    **Sino**
      **Escriba** "Número inválido"
    **F.Si**
  **Fin_Mq**
**Fin_Para**
// Se inicializa en ceros el vector de contadores de números ganadores
**Para**$(i = 1, k, 0)$**Haga**
  $Cont[i] \leftarrow 0$
**Fin_Para**
// Totalizador de los números ganadores de los sorteos
**Para**$(j = 1, k, 1)$**Haga**
  $Num[j] \leftarrow j$
1 2

```
1 2
    Para(i = 1, n, 1)Haga
        Cont[j] ← Cont[j] + A[i, j]
    Fin_Para
  Fin_Para
  // Ordenamiento del vector de número ganadores (Num) y el vector
  // de contadores. Cont, que contiene el número total de veces que
  // un número salió ganador
  i ← 1
  Para(i = 1, k, 1)Haga
    Para(j = i + 1, k, 1)Haga
      Si(Cont[i] < Cont[j])Entonces
        temp ← Cont[i], Cont[i] ← Cont[j], Cont[j] ← temp
        temp ← Num[i], Num[i] ← Num[j], Num[j] ← temp
      F.Si
    Fin_Para
  Fin_Para
  // Impresión de los números que más salieron de las n simulaciones
  Para(i = 1, k, 1)Haga
    Escriba "El número posicionado ", i, " igual a ", Num[i]
    Escriba " y salió ", Cont[i]
  Fin_Para
Fin_proc
```

Ejemplo 4.18 Selector de personal

Una empresa industrial tiene la necesidad de seleccionar $n$ personas para $n$ puestos de trabajo muy parecidos. La empresa tiene $k$ criterios para evaluar cada uno de los candidatos. Estos criterios se evalúan en el intervalo cerrado $[0, 10]$. Por su parte, los $k$ criterios tienen cada uno de ellos asociada una tabla de ponderación ($P$), la cual contiene la importancia del criterio de selección en la escala $[0, 100]$. Construya un algoritmo para seleccionar $n$ personas de los $m$ aspirantes ($m \gg n$) que se presentaron para los puestos de trabajo, por dos metodologías: 1) selección del personal por promedio simple y 2) selección de los empleados utilizando el promedio ponderado.

*Análisis*: El concepto de promedio simple tiene para un conjunto de $x_i$ de $n$ elementos la ecuación de la media representada por

$$\overline{x} = \frac{\displaystyle\sum_{i=1}^{n} x_i}{n}$$

Por su parte, en el concepto de media ponderada para un conjunto $x_i$ de $n$ elementos ($1 \leq i \leq n$), a cada uno de los $n$ elementos de $x_i$ se le asocia un

peso $p_i$ $(1 \leq i \leq n)$, de tal forma que el promedio ponderado está representado por la ecuación matemática

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i p_i}{\sum_{i=1}^{n} p_i}$$

Suponga que el número de criterios que se va a utilizar para evaluar las personas son 5 criterios, cuyos resultados para cada empleado se van a almacenar en el arreglo bidimensional $C[i, j]$. Paralelo a cada fila del arreglo bidimensional se encuentra un vector unidimensional columna con los códigos de los empleados de la empresa industrial. Adicionalmente, en la parte de la derecha se encuentran dos vectores columna: en el primero se almacena los resultados de la media figura 4.18, en el segundo se almacena los resultados de la media ponderada; para cuyo cálculo se utiliza un vector fila de cinco elementos llamado $P$, que contiene la ponderación de cada uno de los criterios, según se presenta en la figura 4.19.

*P*: 5 10 15 20 50

| Código | *C*: | *1* | *2* | *3* | *4* | *5* | Media |
|---|---|---|---|---|---|---|---|
| 1 | | 5 | 5 | 5 | 3 | 5 | **4.6** |
| 2 | | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | | 4 | 4 | 4 | 4 | 4 | 4 |
| 4 | | 1 | 5 | 4 | 5 | 3 | 3.6 |
| 5 | | 4 | 5 | 3 | 5 | 5 | **4.4** |

*Figura 4.18* Selector de personal por el método del promedio simple

Luego, si se va a seleccionar dos personas por el criterio de la media, los empleados seleccionados de los cinco aspirantes son los códigos 1 y 5.

| *P*: | 5 | 10 | 15 | 20 | 50 |
|---|---|---|---|---|---|

| Código | *C*: | *1* | *2* | *3* | *4* | *5* | Media | Ponderada |
|---|---|---|---|---|---|---|---|---|
| 1 | | 5 | 5 | 5 | 3 | 5 | **4.6** | |
| | | 25 | 50 | 75 | 60 | 250 | | **4.60** |
| 2 | | 1 | 1 | 1 | 1 | 1 | 1 | |
| | | 5 | 10 | 15 | 20 | 50 | | 1.00 |
| 3 | | 4 | 4 | 4 | 4 | 4 | 4 | |
| | | 20 | 40 | 60 | 80 | 200 | | 4.00 |
| 4 | | 1 | 5 | 4 | 5 | 3 | 4 | |
| | | 5 | 50 | 60 | 100 | 150 | | **3.65** |
| 5 | | 4 | 5 | 3 | 5 | 5 | **4.4** | |
| | | 20 | 50 | 45 | 100 | 250 | | **4.65** |

*Figura 4.19* Selector de personal por el método de la media ponderada

Por su parte, la selección de los empleados por el criterio de la media ponderada da como resultado el aspirante cinco, con un valor de 4.65; y un segundo, igual a 4.60, correspondiente al primer aspirante.

```
Proc: Selector de personal
| Entero: n, k, m, Codigo[50], i, j, Indmayor
| Real: C[50,50], P[50], Media[50], Pond[50], Sumap, Mayor, temp
| Lea m // Número de aspirantes del proceso de selección
| Lea k // Número de criterios que se va a utilizar en la selección
| Lea n // Número de aspirantes que se va a seleccionar
| // Ciclo repetitivo para que lee el vector de códigos de los aspirantes
| Para(i = 1, m, 1)Haga
| | Lea Codigo[i]
| Fin_Para
| // Ciclo repetitivo que lee los resultados de los criterios (C)
| // de cada aspirante
| Para(i = 1, m, 1)Haga
| | Para(j = 1, k, 1)Haga
| | | Lea C[i, j]
| | Fin_Para
| Fin_Para
| // Ciclo de lectura del vector fila de ponderación
| Para(i = 1, k, 1)Haga
| | Lea P[i]
| Fin_Para
| // Cálculos base para la selección de personal por el método
| // de promedio simple
| Para(i = 1, m, 1)Haga
| | Media[i] ← 0
| | Para(j = 1, k, 1)Haga
| | | Media[i] ← Media[i] + C[i, j]
| | Fin_Para
| | Media[i] ← Media[i]/k
| Fin_Para
| // Cálculos base para la selección de personal por la media ponderada
| // Ciclo que suma el vector fila de los valores ponderados
| Sumap ← 0
| Para(i = 1, k, 1)Haga
| | Sumap ← Sumap + P[i]
| Fin_Para
| Para(i = 1, m, 1)Haga
| | Pond[i] ← 0
| | Para(j = 1, k, 1)Haga
| | | Pond[i] ← Pond[i] + C[i, j] * P[j]
| | Fin_Para
1 2
```

```
1 2
    Pond[i] ← Pond[i]/Sumap
  Fin_Para
  // Estructura lógica de selección de los n aspirantes por la media
  Mayor ← 0, i ← 1

  // Selecciona por mayorización los n aspirantes
  Mq(i ≤ n)Haga
    j ← 1
    // Mayoriza los m aspirantes por la media
    Mq(i ≤ m)Haga
      Si(Media[j] ≥ Mayor)Entonces
        Mayor ← Media[j]
        Indmayor ← j
      F.Si
      j ← j + 1
    Fin_Mq
    // Código del aspirante seleccionado
    Escriba Codigo[Indmayor]
    Media[Indmayor] ← −Media[Indmayor]
    i ← i + 1
    Mayor ← 0
  Fin_Mq

  // Lógica de selección de los n aspirantes para el método de media
  // ponderada. Ordenamiento del vector columna media ponderada
  Para(i = 1, m, 1)Haga
    Para(j = i + 1, m, 1)Haga
      Si(Pond[i] > Pond[j])Entonces
        // Intercambio de promedios ponderados
        temp ← Pond[i]
        Pond[i] ← Pond[j]
        Pond[j] ← temp
        // Intercambio de códigos de los empleados
        temp ← Cod[i]
        Cod[i] ← Cod[j]
        Cod[j] ← temp
      F.Si
    Fin_Para
  Fin_Para
  // Ciclo impresor de los n aspirantes por el método de media ponderada
  Para(i = m, n, −1)Haga
    // Se imprime el código del empleado seleccionado
1 2
```

1 2
**Escriba** $Codigo[i]$
**Fin_Para**
**Fin_proc**

### 4.2.1 Arreglos multidimensionales

Un *arreglo multidimensional* es una variable, con $k$ dimensiones, en la que se almacenan $n_1 \times n_2 \times \cdots \times n_k$ elementos. Para acceder a un elemento de este tipo de arreglos se necesitan $k$ índices diferentes.
Esta clase de arreglos es muy poco usual, debido a que ocupan un gran espacio en la memoria y porque para acceder a un elemento de esta se necesita mucho mas tiempo que el requerido para acceder a uno de tipo unidimensional o bidimensional.

## 4.3 Algoritmos resueltos

Ejemplo 4.19 Arreglo de números aleatorios

Escriba un algoritmo que guarde números aleatorios en un arreglo

$$[1, 2, 3, 4] \implies \boxed{A} \implies [9, 5, 3, 2]$$

*Análisis*: Las variables que se emplean para este algoritmo son un vector $a$, la cantidad de elementos $n$ que lo componen y una variable que sirve de índice para recorrerlo, $j$. Para guardar en un arreglo números aleatorios, lo que se debe hacer primero es leer el tamaño del arreglo, y luego, dentro de un ciclo Para desde 1 hasta el tamaño del arreglo, se le asigna a cada posición del vector un número al azar; para esto se hace uso de la función *random*, que genera valores aleatorios entre uno y el número que se le indique, en este caso 101. Luego solo resta escribir el número.

**Proc**: Números aleatorios
**Entero:** $n$, $j$, $a[50]$
**Escriba** "Digite el tamaño del arreglo: "
**Lea** $n$
**Para**$(j = 1, n, 1)$**Haga**
$a[j] \leftarrow random(101)$
**Escriba** $a[j]$
**Fin_Para**
**Fin_proc**

Ejemplo 4.20 Mover un arreglo una posición a izquierda

Escriba un algoritmo que lea los números de un arreglo y los mueva una posición a la izquierda.

$$[1, 2, 3, 4] \implies \boxed{\mathbf{A}} \implies [2, 3, 4, 1]$$

*Análisis*: Observe que si se desea correr los elementos se puede asignar a cada elemento en la posición $i$ el que esté en la posición $i + 1$. Es decir, $a[i] = a[i + 1]$. Sin embargo, este método resulta exitoso si se realiza de izquierda a derecha. Primero se hace para $i = 1$, es decir, al primer elemento del arreglo se le asigna el segundo elemento; luego, se hace para $i = 2$, es decir, al segundo elemento del vector se le asigna el tercer elemento. Sin embargo, para luego colocar al primer elemento al final, este se debe almacenar en una variable temporal. La variable temporal en este caso es *temp*.

```
Proc: Mover vector a la izquierda
| Entero: n, i, a[50], temp
| Escriba "Digite el tamaño del arreglo: "
| Lea n
| Para(i = 1, n, 1)Haga
| | Escriba "Digite a[", i, "] ="
| | Lea a[i]
| Fin_Para
| temp ← a[1]
| Para(i = 1, n − 1, 1)Haga
| | a[i] ← a[i + 1]
| Fin_Para
| a[n] ← temp
| Para(i = 1, n, 1)Haga
| | Escriba a[i]
| Fin_Para
Fin_proc
```

Ejemplo 4.21 Mover un arreglo una posición a derecha

Escriba un algoritmo que lea los números de un arreglo y los mueva una posición a la derecha.

$$[1, 2, 3, 4] \implies \boxed{\mathbf{A}} \implies [4, 1, 2, 3]$$

*Análisis*: Para mover todos los elementos de un arreglo $a$ una posición hacia la derecha es necesario primero obtener el tamaño del vector, $n$, y los

elementos que contiene, leyéndolos por medio de un ciclo Para desde 1 hasta el tamaño. Como los elementos se correrán hacia la derecha, se necesita guardar el contenido de la última posición, asignándoselo a una variable temporal. Similar a correr los elementos a la izquierda, en este caso resulta exitoso hacerlo solo si se hace de derecha a izquierda. Es decir, primero se actualiza el último elemento del vector, $a[n] = a[n-1]$; posteriormente se hace lo mismo pero para $i = n-1$, es decir, $a[n-1] = a[n-2]$. Finalmente, el primer elemento $i = 1$ se actualiza con la variable *temp*.

```
Proc: Mover vector a la derecha
  Entero: n, j, a[50], temp
  Escriba "Por favor, digite el tamaño del arreglo"
  Lea n
  Para(j = 1, n, 1)Haga
    Escriba "Digite a[", j, "] ="
    Lea a[j]
  Fin_Para
  temp ← a[n]
  Para(i = n, 2, −1)Haga
    a[i] ← a[i − 1]
  Fin_Para
  a[1] ← temp
  Para(j = 1, n, 1)Haga
    Escriba a[j]
  Fin_Para
Fin_proc
```

---

Ejemplo 4.22 Método de búsqueda binaria

Escriba un algoritmo que busque un número en un arreglo usando el método de búsqueda binaria.
Recuerde: el arreglo DEBE estar ordenado. Entrada: $n = 7$

$$[1, 2, 5, 7] \implies \boxed{\mathbf{A}} \implies \text{Se encuentra en la posición 4}$$

*Análisis:* Para implementar el método de búsqueda binaria primero se tiene que leer el tamaño del arreglo $n$ y el elemento *clave* que se va a buscar. Como esta búsqueda exige que los números estén ordenados, entonces, al ir leyendo cada posición del vector se va verificando que el elemento que se vaya a ingresar sea mayor que el último guardado; esto se hace a través de un Para que va desde uno hasta el tamaño del vector; cuando se lee un número comienza a ejecutarse un Haga-Hasta (HH) que se encarga de la verificación. Una vez leído el vector, se ubican 2 banderas: una al principio, *li*, y otra al final, *ls*, y un pivote en la mitad, *pivote*; las banderas se mueven dependiendo de si el elemento que se va a buscar es menor o mayor que el pivote, corriendo

la bandera del final o principio hacia la mitad, respectivamente. Cuando el pivote sea igual es porque se encontró el elemento, y cuando la bandera de inicio sea mayor que la de fin se deja de buscar porque el elemento no se encuentra.

```
Proc: Búsqueda binaria
| Entero: n, clave, j, a[50], li, ls, pivote
| Booleano: sw
| sw ← falso
| Escriba "Digite el tamaño del arreglo y la clave a buscar"
| Lea n, clave
| Escriba "Digite a[1]"
| Lea a[1]
| Para(j = 2, n, 1)Haga
| | sw ← falso
| | HH
| | | Escriba "Digite a[", j, "]"
| | | Lea a[j]
| | | Si(a[j] < a[j − 1])Entonces
| | | | Escriba a[j], "Es menor que", a[j − 1]
| | | | j ← j − 1
| | | Sino
| | | | sw ← verdadero
| | | F.Si
| | Fin_HH(sw = verdadero)
| Fin_Para
|
| li ← 1, ls ← n, sw ← falso
| Mq(li <= ls y sw = falso)Haga
| | pivote ← (li + ls)/2
| | Si(clave = a[pivote])Entonces
| | | sw ← verdadero
| | Sino
| | | Si(clave < a[pivote])Entonces
| | | | ls ← pivote − 1
| | | Sino
| | | | li ← pivote + 1
| | | F.Si
| | F.Si
| Fin_Mq
| Si(sw = verdadero)Entonces
| | Escriba "La clave ", clave, " esta en la posición ", pivote
| Sino
| |
1 2
```

1 2

**Escriba** "La clave no se encuentra"

**F.Si**

**Fin_proc**

---

Ejemplo 4.23 Método de búsqueda secuencial

---

Escriba un algoritmo que busque un número en un arreglo usando búsqueda secuencial. Entrada: $n = 5$

$$[1, 5, 7, 2] \implies \boxed{\textbf{A}} \implies \text{Se encuentra en la posición 2}$$

*Análisis*: En este algoritmo se hace uso de las siguientes variables: un vector de enteros $a$ con su respectivo tamaño $n$, el número *clave* que se va a buscar y un apuntador $j$. La búsqueda secuencial es la más sencilla; simplemente se recorre cada posición del arreglo por medio de un ciclo Mientras que hasta encontrar el elemento que se está buscando o, en su defecto, hasta llegar al final del arreglo sin tener éxito al encontrar el elemento. El ciclo es controlado por una variable boolena $sw$, cuyo valor se altera cuando un número coincide con la clave, con lo cual se detiene la ejecución del ciclo.

**Proc**: Búsqueda secuencial

**Entero:** $n$, $clave$, $j$, $a[50]$

**Booleano:** $sw$

$sw \leftarrow falso$

**Escriba** "Por favor, digite el tamaño del arreglo y la clave que se va a buscar"

**Lea** $n, clave$

**Para**$(j = 1, n, 1)$**Haga**

**Escriba** "Digite $a$[", $j$, "] ="

**Lea** $a[j]$

**Fin_Para**

$j \leftarrow 1$

**Mq**$(j <= n \text{ y } sw = falso)$**Haga**

**Si**$(clave = a[j])$**Entonces**

$sw \leftarrow verdadero$

**F.Si**

$j \leftarrow j + 1$

**Fin_Mq**

**Si**$(sw = verdadero)$**Entonces**

**Escriba** "La clave ", $clave$, " se encuentra en la posición ", $j - 1$

**Sino**

1 2

```
1 2
| | Escriba "Clave errada"
| F.Si
Fin_proc
```

Ejemplo 4.24 Número de veces que aparece el primer término

Escriba un algoritmo que cuente el número de veces que aparece el primer término en un arreglo.

$$[1, 2, 3, 1] \implies \boxed{\textbf{A}} \implies \text{El número 1 está repetido 2 veces}$$

*Análisis*: Primero se declaran las variables necesarias; luego, para contar el número de veces que aparece el primer elemento en un arreglo basta con recorrer cada posición y compararla con el primer elemento, y si es igual, se aumenta un contador *cont*.

```
Proc: Repeticiones del primer elemento
| Entero: n, j, a[50], cont
| cont ← 0
| Escriba "Digite el tamaño del arreglo: "
| Lea n
| Para(j = 1, n, 1)Haga
| | Escriba "Digite a[", j, "] ="
| | Lea a[j]
| Fin_Para
| Para(j = 1, n, 1)Haga
| | Si(a[1] = a[j])Entonces
| | | cont ← cont + 1
| | F.Si
| Fin_Para
| Escriba a[1], " aparece ", cont, " veces"
Fin_proc
```

Ejemplo 4.25 Eliminar elementos duplicados

Realice un algoritmo que lea un arreglo y elimine los elementos duplicados de este.

**Entrada:** Se leerá el tamaño del arreglo; este será generado con números aleatorios
**Salida:** Se eliminarán los números repetidos del arreglo, dejando solo una aparición de estos, y se mostrará este arreglo modificado al usuario.

$$[1, 2, 3, 4, 1] \implies \boxed{\textbf{A}} \implies [1, 2, 3, 4]$$

*Análisis*: Para eliminar los elementos duplicados en un arreglo, lo primero que se tiene que hacer es obtener el tamaño de este y luego llenarlo con elementos, en este caso al azar (*random*); luego se tiene que recorrer cada elemento del arreglo y compararlo con el resto de elementos; y si encuentra un duplicado, simplemente se hace un corrimiento de los elementos siguientes para eliminarlo. El nuevo tamaño del vector es $cn$.

```
Proc: Eliminar duplicados
| Entero: n, cn, i, j, k, a[50]
| Escriba "Por favor, digite el tamaño del arreglo"
| Lea n
| cn ← n
| Escriba "Los valores del arreglo inicial son: "
| Para(j = 1, n, 1)Haga
| | a[j] ← random(11)
| | Escriba a[j]
| Fin_Para
| Escriba "Los valores sin repetición del arreglo son: "
| Para(i = 1, n − 1, 1)Haga
| | Para(j = i + 1, n, 1)Haga
| | | Si(a[i] = a[j])Entonces
| | | | Para(k = j, n − 1, 1)Haga
| | | | | a[k] ← a[k + 1]
| | | | Fin_Para
| | | | cn ← cn − 1
| | | F.Si
| | Fin_Para
| Fin_Para
| Para(j = 1, cn, 1)Haga
| | Escriba a[j]
| Fin_Para
Fin_proc
```

---

Ejemplo 4.26 Ordenamiento por método de Burbuja

Realice un algoritmo que haga el ordenamiento de un vector utilizando el método de burbuja

$$[1, 9, 2, 3, 6] \implies \boxed{\textbf{A}} \implies \text{El arreglo ordenado: } [1, 2, 3, 6, 9]$$

*Análisis*: El ordenamiento a través del método burbuja es el más sencillo. En este, primero se lee el arreglo y luego se toma el elemento en la primera posición del arreglo y se compara esa posición con cada una de las posiciones siguientes, y cada vez que encuentre una menor se intercambia. Una vez se haya comparado la primera posición con el resto, se continúa con las siguientes, y progresivamente el arreglo se ordenará.

**Proc**: Método burbuja
  **Entero:** $n$, $j$, $h$, $temp$, $a[50]$
  **Escriba** "Digite el tamaño del arreglo"
  **Lea** $n$
  **Para**$(j = 1, n, 1)$**Haga**
    **Escriba** "Digite el valor de $a$[", $j$, "]"
    **Lea** $a[j]$
  **Fin_Para**
  **Escriba** "El arreglo sin ordenar es el siguiente"
  **Para**$(j = 1, n, 1)$**Haga**
    **Escriba** $a[j]$," "
  **Fin_Para**
  **Para**$(j = 1, n - 1, 1)$**Haga**
    **Para**$(h = j + 1, n, 1)$**Haga**
      **Si**$(a[h] < a[j])$**Entonces**
        $temp \leftarrow a[h]$
        $a[h] \leftarrow a[j]$
        $a[j] \leftarrow temp$
      **F.Si**
    **Fin_Para**
  **Fin_Para**
  **Escriba** "El arreglo ordenado es el siguiente:"
  **Para**$(j = 1, n, 1)$**Haga**
    **Escriba** $a[j]$
  **Fin_Para**
**Fin_proc**

---

Ejemplo 4.27 Triángulo inferior por filas

Escriba un algoritmo que muestre un triángulo inferior acotado por la diagonal principal por filas.

$$3 \Longrightarrow \boxed{\mathbf{A}} \Longrightarrow \begin{array}{lll} 1 & & \\ 2 & 3 & \\ 4 & 5 & 6 \end{array}$$

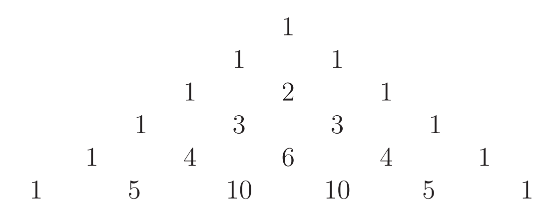
*Análisis*: Las variables $i$ y $j$ serán los índices para recorrer la matriz; las variables $m$ y $n$ son el número de filas y de columnas de la matriz, respectivamente; en este caso la matriz es cuadrada, por lo tanto, $m = n$. Luego se llena la matriz con ceros en todas sus posiciones. En este caso la matriz debe llenarse como un triángulo acotado inferiormente por la diagonal principal, por lo tanto, la variable $j$ va desde 1 hasta $i$ y va generando la forma deseada. Finalmente se escribe el contenido de la matriz.

**Proc**: Triángulo inferior
  **Entero:** $n$, $m$, $k$, $i$, $j$, $a[50, 50]$
  $k \leftarrow 1$
  **Escriba** "Digite el número de filas: "
  **Lea** $m$
  $n \leftarrow m$
  **Para**$(i = 1, m, 1)$**Haga**
    **Para**$(j = 1, n, 1)$**Haga**
      $a[i, j] \leftarrow 0$
    **Fin_Para**
  **Fin_Para**
  **Para**$(i = 1, m, 1)$**Haga**
    **Para**$(j = 1, i, 1)$**Haga**
      $a[i, j] \leftarrow k$
      $k \leftarrow k + 1$
    **Fin_Para**
  **Fin_Para**
  **Para**$(i = 1, m, 1)$**Haga**
    **Para**$(j = 1, n, 1)$**Haga**
      **Si**$(a[i, j] \neq 0)$**Entonces**
        **Escriba** $a[i, j]$
      **F.Si**
    **Fin_Para**
  **Fin_Para**
**Fin_proc**

---

Ejemplo 4.28 Triángulo de Pascal de $N$ filas

Diseñe un algoritmo que genere el triángulo de $N$ filas.




*Análisis*: La instrucción 'Escriba "\n"' significa pasar a la siguiente línea en la salida, es como un "enter". Primero se declaran las variables que se van a utilizar; luego se obtiene el número de filas y se halla el número de columnas mediante la fórmula $2m - 1$; posteriormente se rellena toda la matriz con 0 y desde la mitad superior se comienza a llenar la matriz utilizando las diagonales y sumando los elementos anteriores. Finalmente se escribe la matriz, cambiando los 0 por los espacios vacíos.

```
Proc: Triángulo de N filas
| Entero: n, m, i, j, a[50, 50]
| Escriba "Digite el número de filas: "
| Lea m
| n ← 2 * m − 1
| Escriba "Triángulo de Pascal"
| Para(i = 1, m, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | a[i, j] ← 0
| | Fin_Para
| Fin_Para
| a[1, m] ← 1
| Para(i = 2, m, 1)Haga
| | Para(j = m + 1 − i, n − 1, 1)Haga
| | | a[i, j] ← a[i − 1, j − 1] + a[i − 1, j + 1]
| | Fin_Para
| Fin_Para
| a[m, n] ← a[m − 1, n − 1]
| Para(i = 1, m, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | Si(a[i, j] = 0)Entonces
| | | | Escriba " "
| | | Sino
| | | | Escriba a[i, j]
| | | F.Si
| | Fin_Para
| | Escriba "\n"
| Fin_Para
Fin_proc
```

Ejemplo 4.29 Triángulo inferior por columnas

Escriba un algoritmo que muestre un triángulo inferior acotado por la diagonal principal por columnas.

$$3 \Longrightarrow \boxed{\mathbf{A}} \Longrightarrow \begin{matrix} 1 & & \\ 2 & 4 & \\ 3 & 5 & 6 \end{matrix}$$

*Análisis*: Este algoritmo es similar al descrito anteriormente, sus declaraciones y captura de valores son iguales, pero en este caso se llena la matriz por columnas, ya que los índices de la matriz son cambiados de $[i, j]$ a $[j, i]$.

```
Proc: Triángulo inferior
| Entero: n, m, i, j, k, a[50, 50]
| k ← 1
| Escriba "Digite el número de filas: "
| Lea m
| n ← m
| Para(i = 1, m, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | a[i, j] ← 0
| | Fin_Para
| Fin_Para
| Para(i = 1, m, 1)Haga
| | Para(j = 1, i, 1)Haga
| | | a[j, i] ← k
| | | k ← k + 1
| | Fin_Para
| Fin_Para
| Para(i = 1, m, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | Si(a[i, j] ≠ 0)Entonces
| | | | Escriba a[i, j]
| | | F.Si
| | Fin_Para
| Fin_Para
Fin_proc
```

---

Ejemplo 4.30 Triángulo superior

Escriba un algoritmo que muestre un triángulo superior acotado por la diagonal secundaria por filas.

$$3 \implies \boxed{\mathbf{A}} \implies \begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & \\ 6 & & \end{array}$$

*Análisis*: Las variables $i$ y $j$ serán los índices para recorrer la matriz; las variables $m$ y $n$ son el número de filas y de columnas de la matriz, respectivamente; en este caso la matriz es cuadrada, por lo tanto $m = n$. Luego se llena la matriz con ceros en todas sus posiciones. En este caso el triángulo es superior y es acotado por la diagonal secundaria de la matriz, por lo tanto, la variable $j$ va desde 1 hasta $(n - i) + 1$, que es la diagonal secundaria de toda matriz cuadrada. Finalmente se escribe el contenido de la matriz.

```
Proc: Triángulo superior
  Entero: n, m, i, j, k, a[50, 50]
  k ← 1
  Escriba "Digite el número de filas: "
  Lea m
  n ← m
  Para(i = 1, m, 1)Haga
    Para(j = 1, n, 1)Haga
      a[i, j] ← 0
    Fin_Para
  Fin_Para
  Para(i = 1, m, 1)Haga
    Para(j = 1, (n − i) + 1, 1)Haga
      a[i, j] ← k
      k ← k + 1
    Fin_Para
  Fin_Para
  Para(i = 1, m, 1)Haga
    Para(j = 1, n, 1)Haga
      Si(a[i, j] ≠ 0)Entonces
        Escriba a[i, j]
      F.Si
    Fin_Para
  Fin_Para
Fin_proc
```

Ejemplo 4.31 Triángulo acotado invertido

Escriba un algoritmo que muestre un triángulo acotado invertido; la entrada es el número de columnas; el número de filas está relacionado con el número de columnas así: $n = 2 * m - 1$.

$$3 \implies \boxed{\mathbf{A}} \implies \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ & 6 & 7 & 8 & \\ & & 9 & & \end{array}$$

*Análisis*: Las variables $i$ y $j$ serán los índices para recorrer la matriz; las variables $m$ y $n$ son el número de filas y de columnas de la matriz, respectivamente; en este caso, a $n$ se le asigna el valor de $2m - 1$ (tómese un minuto para pensar el porqué de la afirmación). Luego se llena la matriz con ceros en todas sus posiciones. En este caso este triángulo es acotado a ambos lados, por lo tanto, la variable $j$ comienza en el valor de la variable $i$ y va hasta $n + 1 - i$. Finalmente se escribe el contenido de la matriz.

```
Proc: Triángulo acotado invertido
| Entero: n, m, i, j, k, a[50,50]
| k ← 1
| Escriba "Digite el número de filas: "
| Lea m
| n ← 2 * m − 1
| Para(i = 1, m, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | a[i][j] ← 0
| | Fin_Para
| Fin_Para
| Para(i = 1, m, 1)Haga
| | Para(j = i, n + 1 − i, 1)Haga
| | | a[i][j] ← k
| | | k ← k + 1
| | Fin_Para
| Fin_Para
| Para(i = 1, m, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | Si(a[i][j] ≠ 0)Entonces
| | | | Escriba a[i][j]
| | | F.Si
| | Fin_Para
| Fin_Para
Fin_proc
```

Ejemplo 4.32 Matriz acotada por la función valor absoluto

Realice un algoritmo que muestre una matriz acotada con la forma de la gráfica de la función de valor absoluto; tal y como muestra el ejemplo, la entrada es el número de columnas.

$$3 \Longrightarrow \boxed{\textbf{A}} \Longrightarrow \begin{array}{ccc} & & 1 \\ & 2 & 3 \\ 4 & 5 & 6 \\ & 7 & 8 \\ & & 9 \end{array}$$

*Análisis*: Como primer paso se obtiene el número de columnas, y hallamos el número de filas por la fórmula $2m - 1$ (tómese un instante de tiempo para pensar el porqué de la afirmación); después se rellena toda la matriz con 0. Posteriormente llenamos la matriz recorriendo la matriz acotada por la función valor absoluto sobre $n - i$ y sumando 1. Finalmente escribimos la matriz, cambiando los 0 por espacios vacíos.

```
Proc: Matriz acotada por la función valor absoluto
| Entero: n, m, i, j, k, a[50,50]
| k ← 1
| Escriba "Digite el número de columnas: "
| Lea n
| m ← 2 * n − 1
| Para(i = 1, m, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | a[i, j] ← 0
| | Fin_Para
| Fin_Para
| Para(i = 1, m, 1)Haga
| | Para(j = abs(n − i) + 1, n, 1)Haga
| | | a[i, j] ← k
| | | k ← k + 1
| | Fin_Para
| Fin_Para
| Para(i = 1, m, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | Si(a[i, j] = 0)Entonces
| | | | Escriba " "
| | | Sino
| | | | Escriba a[i, j]
| | | F.Si
| | Fin_Para
| Fin_Para
Fin_proc
```

Ejemplo 4.33 Triángulo inferior acotado por la diagonal secundaria

Escriba un algoritmo que muestre un triángulo inferior acotado por la diagonal secundaria por filas.

$$3 \implies \boxed{\mathbf{A}} \implies \begin{matrix} & & 1 \\ & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

*Análisis*: Este algoritmo es similar al descrito anteriormente, su captura de valores es igual, sin embargo, en este caso se llena la matriz por filas, y el valor inicial y final de la variable $j$ son modificados.

```
Proc: Triángulo inferior acotado por diagonal secundaria
| Entero: n, m, i, j, k, a[50,50]
| k ← 1
1
```

```
1
    Escriba "Digite el número de filas: "
    Lea m
    n ← m
    Para(i = 1, m, 1)Haga
        Para(j = 1, n, 1)Haga
            a[i, j] ← 0
        Fin_Para
    Fin_Para
    Para(i = 1, m, 1)Haga
        Para(j = (n − i) + 1, n, 1)Haga
            a[i, j] ← k
            k ← k + 1
        Fin_Para
    Fin_Para
    Para(i = 1, m, 1)Haga
        Para(j = 1, n, 1)Haga
            Si(a[i, j] ≠ 0)Entonces
                Escriba a[i, j]
            F.Si
        Fin_Para
    Fin_Para
Fin_proc
```

Ejemplo 4.34 Recorrido de zigzag I

Escriba un algoritmo que realice el recorrido de zigzag empezando hacia arriba y la derecha.

3 $\Longrightarrow$ **A** $\Longrightarrow$

| | | |
|---|---|---|
| 1 | 2 | 6 |
| 3 | 5 | 7 |
| 4 | 8 | 9 |

*Análisis*: Como primer paso se procede a recorrer la matriz $M \times N$ veces. Si la suma de los apuntadores módulo 2 es cero, entonces se mueve una posición hacia abajo y una hacia la izquierda; si no, se mueve una posición hacia arriba y una hacia la derecha. Hay que tener en cuenta que si los apuntadores llegan a ser cero o si las filas o columnas llegan a ser mayores que el tamaño declarado $(M, N)$, estos deben ser reubicados, dependiendo del lugar donde se encuentren. Para finalizar se escribe la matriz resultante.

```
Proc: Recorrido zigzag
| Entero: n, m, i, j, k, a[50,50]
| Escriba "Digite número de filas: "
| Lea m
| Escriba "Digite número de columnas: "
| Lea n
| Para(i = 1, m, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | a[i, j] ← 0
| | Fin_Para
| Fin_Para
| i ← 2
| j ← 0
| Para(k = 1, m * n, 1)Haga
| | Si((i + j) mod 2 = 0)Entonces
| | | i ← i − 1
| | | j ← j + 1
| | Sino
| | | i ← i + 1
| | | j ← j − 1
| | F.Si
| | Si(j = n + 1)Entonces
| | | i ← i + 2
| | | j ← n
| | F.Si
| | Si(i = m + 1)Entonces
| | | j ← j + 2
| | | i ← m
| | F.Si
| | Si(i = 0)Entonces
| | | i ← 1
| | F.Si
| | Si(j = 0)Entonces
| | | j ← 1
| | F.Si
| | a[i, j] ← k
| Fin_Para
| Para(i = 1, m, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | Escriba a[i, j], " "
| | Fin_Para
| Fin_Para
Fin_proc
```

Ejemplo 4.35 Recorrido de zigzag II

Escriba un algoritmo que realice el recorrido de zigzag empezando hacia abajo y a la izquierda.

$3 \times 3 \implies \boxed{\mathbf{A}} \implies$

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 2 | 5 | 8 |
| 6 | 7 | 9 |

*Análisis*: El ejercicio es similar al anterior, pero el recorrido lo hace hacia abajo y a la izquierda, y se modifican los índices $i$ y $j$ para cambiar la dirección.

**Proc**: Recorrido alterno de zigzag
- **Entero:** $n$, $m$, $i$, $j$, $k$, $a[50, 50]$
- **Escriba** "Digite número de filas: "
- **Lea** $m$
- **Escriba** "Digite número de columnas: "
- **Lea** $n$
- **Para**$(i = 1, m, 1)$**Haga**
  - **Para**$(j = 1, n, 1)$**Haga**
    - $a[i, j] \leftarrow 0$
  - **Fin_Para**
- **Fin_Para**
- $i \leftarrow 0$
- $j \leftarrow 2$
- **Para**$(k = 1, m * n, 1)$**Haga**
  - **Si**$((i + j)$ **mod** $2 = 0)$**Entonces**
    - $i \leftarrow i + 1$
    - $j \leftarrow j - 1$
  - **Sino**
    - $i \leftarrow i - 1$
    - $j \leftarrow j + 1$
  - **F.Si**
  - **Si**$(j = n + 1)$**Entonces**
    - $i \leftarrow i + 2$
    - $j \leftarrow n$
  - **F.Si**
  - **Si**$(i = m + 1)$**Entonces**
    - $j \leftarrow j + 2$
    - $i \leftarrow m$
  - **F.Si**

1 2

1 2
**Si**$(i = 0)$**Entonces**
$i \leftarrow 1$
**F.Si**
**Si**$(j = 0)$**Entonces**
$j \leftarrow 1$
**F.Si**
$a[i][j] \leftarrow k$
**Fin_Para**
**Para**$(i = 1, m, 1)$**Haga**
**Para**$(j = 1, n, 1)$**Haga**
**Escriba** $a[i, j]$, " "
**Fin_Para**
**Fin_Para**
**Fin_proc**

---

Ejemplo 4.36 Mezcla de dos vectores

Realice un algoritmo que genere un vector a partir de dos vectores dados, y que contenga todos los elementos de ambos ordenados de manera ascendente.

$$\begin{array}{c} [1,9,2,3] \Longrightarrow \\ [1,4,2] \Longrightarrow \end{array} \boxed{\mathbf{A}} \Longrightarrow [1,1,2,2,3,4,9]$$

*Análisis*: Se creará un nuevo vector, y el tamaño de este será la suma de los tamaños de los dos vectores leídos. Se procede a crear los vectores $a$ y $b$, para luego organizarlos; la lógica de organización se basa en ir comparando las posiciones de los dos vectores e ir agregando los elementos según el criterio; al agregar un elemento en el vector nuevo se aumenta el apuntador de la posición correspondiente al vector del cual se tomó el elemento añadido.

**Proc**: Generar vector
**Entero:** $n1$, $n2$, $n3$, $i$, $j$, $j1$, $j2$, $j3$, $cont$, $temp$, $a[50]$, $b[50]$, $c[50]$
**Escriba** "Escriba el tamaño del arreglo 1: "
**Lea** $n1$
**Escriba** "Escriba el tamaño del arreglo 2: "
**Lea** $n2$
$n3 \leftarrow n1 + n2$
$a[1] \leftarrow 1 + random(100)$
**Escriba** "Arreglo 1"
**Escriba** $a[1]$
// Ordenar vector $a$
**Para**$(i = 2, n1, 1)$**Haga**
$a[i] \leftarrow 1 + random(100)$
1 2

```
1 2
    Para(j = i, 2, −1)Haga
      Si(a[j] < a[j − 1])Entonces
        temp ← a[j]
        a[j] ← a[j − 1]
        a[j − 1] ← temp
      Sino
        j ← 1
      F.Si
    Fin_Para
    Escriba ",", a[i]
  Fin_Para
  Escriba "\n"

  b[1] ← 1 + random(100)
  Escriba "Arreglo 2: "
  Escriba b[1]
  Para(i = 2, n2, 1)Haga
    b[i] ← 1 + random(100)
    Para(j = i, 2, −1)Haga
      Si(b[j] < b[j − 1])Entonces
        temp ← b[j]
        b[j] ← b[j − 1]
        b[j − 1] ← temp
      Sino
        j ← 1
      F.Si
    Fin_Para
    Escriba ",", b[i]
  Fin_Para
  Escriba "\n"

  // Se combinan los vectores a y b y se ordenan
  j1 ← 1
  j2 ← 1
  j3 ← 1
  Mq(j1 <= n1 y j2 <= n2)Haga
    Si(a[j1] < b[j2])Entonces
      c[j3] ← a[j1]
      j1 ← j1 + 1
    Sino
      c[j3] ← b[j2]
      j2 ← j2 + 1
    F.Si
1 2
```

```
1 2
| | j3 ← j3 + 1
| Fin_Mq
| Si(j2 = n2 + 1)Entonces
| | Para(cont = j3, n3, 1)Haga
| | | c[cont] ← a[j1]
| | | j1 ← j1 + 1
| | Fin_Para
| Sino
| | Para(cont = j3, n3, 1)Haga
| | | c[cont] ← b[j2]
| | | j2 ← j2 + 1
| | Fin_Para
| F.Si
| Escriba "Arreglo 3: "
| Para(j = 1, n3, 1)Haga
| | Escriba c[j], " "
| Fin_Para
Fin_proc
```

Ejemplo 4.37 Cuadrado mágico

Realice un algoritmo que muestre un cuadrado mágico con las dimensiones leídas. Recuerde que en el cuadrado mágico el número de todas las filas, columnas y diagonales da como resultado el mismo número.

*Análisis*: El algoritmo debe recibir como entrada un número entero positivo impar $n$ y como salida un cuadrado mágico con dichas dimensiones. Por ejemplo,

3 $\Longrightarrow$ **A** $\Longrightarrow$

| | | |
|---|---|---|
| 8 | 1 | 6 |
| 3 | 5 | 7 |
| 4 | 9 | 2 |

Para construir un cuadrado mágico se deben seguir los siguientes pasos:

1. Coloque el número 1 en la casilla del medio de la fila superior.
2. Para colocar el número siguiente desplácese una casilla arriba y una hacia la derecha. Si el número que intenta colocar queda fuera del cuadrado, imagine que todos los bordes están unidos. Continúe aplicando esto hasta que se encuentre con que no puede colocar el número correspondiente porque en la casilla ya hay un número.
3. Si al intentar colocar un número se da cuenta de que la casilla en que debe ir ya está ocupada, entonces colóquelo en la casilla que está inmediatamente debajo del último número que colocó.

4. Complete el cuadrado aplicando los dos pasos anteriores.

La lógica que debe seguir el algoritmo es la siguiente: se coloca el número 1 en la casilla del medio de la fila superior; para colocar el número siguiente hay que desplazarse una casilla arriba y una hacia la derecha. Si el número que intenta colocar queda fuera del cuadrado, imagine que todos los bordes están unidos. Se continúa aplicando esta lógica hasta que se encuentra con que no puede colocar el número correspondiente porque en la casilla ya existe uno; al existir un número se debe colocar este en la casilla que está inmediatamente debajo del último número que se colocó. Finalmente se completa el cuadrado aplicando el procedimiento anterior.

**Proc**: Cuadrado mágico
  **Entero:** $n$, $m$, $i$, $j$, $k$, $auxi$, $auxj$, $a[50, 50]$
  **Booleano:** $sw$
  $k \leftarrow 0$
  $sw \leftarrow falso$
  **Escriba** "Digite número de filas y columnas (impares) $\longrightarrow$"
  **Lea** $m, n$
  **Para**$(i = 1, m, 1)$**Haga**
    **Para**$(j = 1, n, 1)$**Haga**
      $a[i, j] \leftarrow 0$
    **Fin_Para**
  **Fin_Para**

  $i \leftarrow 1$
  $j \leftarrow (n + 1)/2$
  $a[i, j] \leftarrow 1$
  **Para**$(k = 2, n * n, 1)$**Haga**
    $i \leftarrow i - 1$
    $j \leftarrow j + 1$
    $auxi \leftarrow i$ **mod** $n$
    $auxj \leftarrow j$ **mod** $n$
    **Si**$(auxi = 0)$**Entonces**
      $auxi \leftarrow n$
    **F.Si**
    **Si**$(auxj = 0)$**Entonces**
      $auxj \leftarrow n$
    **F.Si**
    **Si**$(a[auxi, auxj] \neq 0)$**Entonces**
      $a[auxi, auxj] \leftarrow k$
    **Sino**
      $i \leftarrow i + 2$
      $j \leftarrow j - 1$
1 2 3

```
1 2 3
| | | auxi ← i mod n
| | | auxj ← j mod n
| | | Si(auxi = 0)Entonces
| | | | auxi ← n
| | | F.Si
| | | Si(auxj = 0)Entonces
| | | | auxj ← n
| | | F.Si
| | | a[auxi, auxj] ← k
| | F.Si
| Fin_Para
| Escriba "Cuadrado Mágico"
| Para(i = 1, m, 1)Haga
| | Para(j = 1, n, 1)Haga
| | | Escriba a[i, j]
| | Fin_Para
| Fin_Para
Fin_proc
```

Ejemplo 4.38 Criba de Eratóstenes

Construya un algoritmo que permita determinar el número de números primos entre 1 y un número ingresado por el usuario a través de la criba de Eratóstenes. Asuma $n > 10000$.

*Análisis*: Una de las formas más eficientes de generar números primos es a través de la criba de Eratóstenes. Observe que otra forma es ir revisando de número en número si es primo o no, sin embargo, resulta muy ineficiente comparado con una criba de Eratóstenes (piense ¿por qué?).

Para generar números primos a través de la criba de Eratóstenes se forma un arreglo de todos los números naturales comprendidos entre 2 y $n$. En ella se van tachando todos los números que no son primos de la siguiente forma: comenzado por el 2, se tachan todos sus múltiplos (es decir, 4, 6, 8, 10, ...); comenzando nuevamente se busca un número que no haya sido tachado, es decir, 3, y se tachan todos sus múltiplos (es decir, 6, 9, 12, ...); nuevamente se busca un número que no haya sido tachado antes, es decir, 5, y se tachan todos sus múltiplos. El proceso culmina cuando el cuadrado del mayor número primo confirmado es mayor que $n$ (piense ¿por qué?).

El proceso anterior se puede resumir en:

1. Haga una lista de todos los números de 2 a $n$.
2. Encuentre el siguiente número $p$ que no haya sido tachado. Este es un primo. Si es mayor que $\sqrt{n}$, vaya al paso 5.

3. Tache todos los múltiplos de $p$ que no han sido tachados aún.
4. Vaya al paso 2.
5. Los números no tachados son los primos menores que $n$.

```
Proc: Criba de Eratóstenes
|  Entero: n, lim, i, j, c, criba[50]
|  Escriba "Ingrese el valor de n: "
|  Lea n
|  lim ← ⌊√n⌋
|  // Se inicia la criba en 'falso'
|  Para(i = 2, n, 1)Haga
|  |  criba[i] ← 0
|  Fin_Para
|  // Se marcan todos los números pares mayores que 2
|  Para(i = 4, 2, n)Haga
|  |  criba[i] ← 1
|  Fin_Para
|  Para(i = 3, lim, 2)Haga
|  |  Si(criba[i] = 0)Entonces
|  |  |  Para(j = i * i, lim, 2 * i)Haga
|  |  |  |  criba[j] ← 1
|  |  |  Fin_Para
|  |  F.Si
|  Fin_Para
|  c ← 0
|  Para(i = 2, n, 1)Haga
|  |  Si(criba[i] = 0)Entonces
|  |  |  c ← c + 1
|  |  F.Si
|  Fin_Para
|  Escriba "El número de números primos es: ", c
Fin_proc
```

El anterior algoritmo se puede optimizar si se ignoran los números pares y se crea la criba solamente para números impares. Si se ignoran los números pares, se ocupa solamente la mitad de la memoria, dado que en vez de crear un arreglo de tamaño $n$ se crea un arreglo de tamaño $\frac{1}{2}(n-2)$, ignorando el 1 y todos los números pares. Sin embargo, ahorrar dicho espacio requiere un cambio en los índices usados. En este caso el $i$-ésimo término del vector corresponde con el número impar $2i+1$. Por lo tanto, si $p = 2i+1$, se tiene que $p^2 = 4i^2 + 4i + 1$ tiene el índice $2i(i+1)$. Para esto vea que

$$p^2 = 4i^2 + 4i + 1 = 2(2i(i+1)) + 1.$$

Por otro lado, si $m = kp$ corresponde a $j$, entonces $m + 2p$ corresponde a $j + p$. Por lo tanto, si el $i$-ésimo índice no ha sido tachado, el ciclo interior inicia en $2i(i + 1)$ y procede con paso $2i + 1$. El código anterior se convierte en:

```
Proc: Criba de Eratóstenes
| Entero: n, lim, i, j, c, criba[50]
| Escriba "Ingrese el valor de n: "
| Lea n
| lim ← (⌊√n⌋ − 1)/2
| n ← (n − 1)/2
| // Se inicia la criba en 'falso'
| Para(i = 2, n, 1)Haga
| | criba[i] ← 0
| Fin_Para
| // Se marcan todos los números pares mayores que 2
| Para(i = 4, 2, n)Haga
| | criba[i] ← 1
| Fin_Para
| Para(i = 3, lim, 2)Haga
| | Si(criba[i] = 0)Entonces
| | | Para(j = 2 ∗ i ∗ (i + 1), lim, 2 ∗ i + 1)Haga
| | | | criba[j] ← 1
| | | Fin_Para
| | F.Si
| Fin_Para
| c ← 1 // 2 es primo
| Para(i = 2, n, 1)Haga
| | Si(criba[i] = 0)Entonces
| | | c ← c + 1
| | F.Si
| Fin_Para
| Escriba "El número de números primos es: ", c
Fin_proc
```

## 4.4 Ejercicios propuestos

1. Realice un algoritmo que lea un vector de $N$ valores enteros y, sin usar vectores auxiliares, invierta el contenido de este y muéstrelo.

   **Entrada:** $A = [4, 2, 6, 3, 7, 2]$.

   **Salida:** El vector invertido es $[2, 7, 3, 6, 2, 4]$.

2. Dado una serie de notas de alumnos, almacénelas en un vector, calcule la media y determine cuántos alumnos superan la media y cuántos están por debajo de esta.

**Entrada:** $A = [4.4, 2.2, 4.6, 3.1, 5.0, 2.1]$. **Salida:** 3 notas que superan la media; 3 notas están por debajo de esta.

3. Diseñe un algoritmo que reciba como datos de entrada un número entero $n$, $n$ números enteros, que serán los elementos de un vector de tamaño $n$ y un número entero $x$. Este algoritmo debe regresar como dato de salida la posición dentro del vector en la que se encuentra $x$. Esta posición debe ser determinada usando el método de búsqueda Binaria.

   **Entrada:** $n = 6$, $A = [4, 2, 6, 3, 7, 2]$, $x = 4$.

   **Salida:** El número SI se encuentra en el arreglo.

4. Realice el inciso anterior con Búsqueda Secuencial.

5. Elabore un algoritmo que reciba como datos de entrada un número entero positivo $n$ y los $n$ elementos de un vector de tamaño $n$, y que regrese como dato de salida cuántas veces se repite el último elemento del vector.

   **Entrada:** $n = 6$, $A = [4, 2, 5, 3, 7, 5]$.

   **Salida:** El número 5 se encuentra repetido 2 veces.

6. Dado un array unidimensional y un número, muestre los elementos que son mayores que este número.

   **Entrada:** $A = [4, 2, 6, 3, 7, 2]$, $x = 4$.

   **Salida:** Los elementos mayores que 4 son 6,7.

7. Muestre el mínimo elemento de un array dado.
   **Entrada:** $A = [4, 2, 6, 3, 7, 2]$.

   **Salida:** El elemento mínimo es 2.

8. Dado un vector de enteros, realice un algoritmo que ordene sus elementos descendentemente.

   **Entrada:** $A = [4, 2, 6, 3, 7, 2]$.

   **Salida:** El vector ordenado descendentemente es $[7, 6, 4, 3, 2, 2]$.

9. Escriba un algoritmo que indique si dos vectores de enteros son "circularmente iguales", que significa tener los mismos elementos y en el mismo orden relativo circularmente.

   **Entrada:** $A = [1, 2, 3, 4]$, $B = [3, 4, 1, 2]$.

   **Salida:** Los dos vectores SI son circularmente iguales.

10. Dado un vector de enteros, realice un algoritmo que diga cuál es la subcadena más larga de números secuenciales creada con los enteros.

    **Entrada:** $A = [4, 1, 3, 5, 2, 3, 4, 5, 2, 4]$.

    **Salida:** La subcadena más larga es 2,3,4,5.

11. Escriba un algoritmo que calcule la suma de los elementos de la diagonal principal de una matriz $n \times n$.

**Entrada:**

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 1 | 6 | 7 |
| 8 | 9 | 1 | 1 |
| 2 | 3 | 4 | 1 |

**Salida:** La suma de los elementos de la diagonal principal es 4.

12. Escriba un algoritmo que permita sumar el número de elementos positivos y negativos de una matriz de $m \times n$.

**Entrada:**

| | | | |
|---|---|---|---|
| $-1$ | 2 | 3 | 4 |
| 5 | $-1$ | 6 | 7 |
| 8 | 9 | $-1$ | 1 |
| 2 | 3 | 4 | $-1$ |

**Salida:** La suma de los Números Positivos es 54.
La suma de los Números Negativos es $-4$.

13. Dada una matriz $m \times n$, realice un algoritmo para voltearla de izquierda a derecha.

**Entrada:**

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |

**Salida:** La matriz volteada de izquierda a derecha es

| | | |
|---|---|---|
| 3 | 2 | 1 |
| 6 | 5 | 4 |
| 9 | 8 | 7 |
| 12 | 11 | 10 |

14. Repita el ejercicio anterior para voltear la matriz de arriba hacia abajo.

15. Una matriz $m \times m$ es simétrica si sus elementos satisfacen $a_{ij} = a_{ji}$. Realice un algoritmo que determine si una matriz es o no simétrica.

**Entrada:**

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 2 | 1 | 2 | 3 |
| 3 | 2 | 1 | 2 |
| 4 | 3 | 2 | 1 |

**Salida:** La matriz SI es simétrica.

16. Realice un algoritmo que intercambie las filas $i$ y $j$ de una matriz $m \times n$.

**Entrada:** $i = 1$, $j = 2$,

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 1 | 6 | 7 |
| 8 | 9 | 1 | 1 |
| 2 | 3 | 4 | 1 |

**Salida:**

| | | | |
|---|---|---|---|
| 5 | 1 | 6 | 7 |
| 1 | 2 | 3 | 4 |
| 8 | 9 | 1 | 1 |
| 2 | 3 | 4 | 1 |

17. Escriba un algoritmo para ordenar una matriz de tamaño $n \times m$ de menor a mayor con el siguiente proceso: guarde primero por filas, de la 1 a la $N$, la matriz en un vector de tamaño $mn$ y luego ordene el vector. Finalmente copie de nuevo el vector en la matriz.

**Entrada:**

| | | | |
|---|---|---|---|
| 5 | 1 | 6 | 7 |
| 1 | 2 | 3 | 4 |
| 8 | 9 | 1 | 1 |
| 2 | 3 | 4 | 1 |

**Salida:**

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 3 |
| 3 | 4 | 4 | 5 |
| 6 | 7 | 8 | 9 |

18. Una matriz tiene un punto de silla en una de sus componentes si este es el mayor valor de su columna y el menor de su fila. Diseñe un algoritmo que recibiendo una matriz $n \times n$ muestre las coordenadas de todos sus puntos silla.

**Entrada:**

| | | | | |
|---|---|---|---|---|
| 5 | 3 | 3 | 8 | 6 |
| 10 | 3 | 2 | 5 | 7 |
| 7 | 7 | 6 | 8 | 10 |
| 4 | 17 | 5 | 9 | 0 |
| 1 | 3 | 4 | 7 | 0 |

**Salida:** La matriz tiene 1 punto de silla, y se encuentra en la posición $(3, 3)$.

19. Realice un algoritmo que dada una matriz de tamaño $2n * 2n$ intercambie el primer cuadrante con el tercero y el segundo cuadrante con el cuarto.

**Entrada:** $i = 2$.

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 2 |
| 1 | 1 | 2 | 2 |
| 3 | 3 | 4 | 4 |
| 3 | 3 | 4 | 4 |

**Salida:**

| | | | |
|---|---|---|---|
| 4 | 4 | 3 | 3 |
| 4 | 4 | 3 | 3 |
| 2 | 2 | 1 | 1 |
| 2 | 2 | 1 | 1 |

20. Realice un algoritmo que dada una matriz de tamaño $n \times n$ encuentre y muestre solo los elementos que no se repiten.

**Entrada:**

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 3 |
| 5 | 4 | 3 | 3 |
| 7 | 7 | 4 | 9 |
| 8 | 8 | 7 | 8 |

**Salida:** Los elementos que no se repiten son 2, 5, 9.

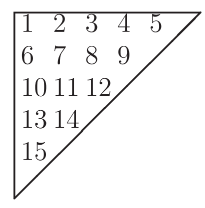
21. Iniciando en la posición central con un movimiento de espiral en sentido contrario a las manecillas del reloj se puede formar un cuadrado de lado siete, como el que se muestra a continuación:

| | | | | | | |
|---|---|---|---|---|---|---|
| ***37*** | 36 | 35 | 34 | 33 | 32 | ***31*** |
| 38 | ***17*** | 16 | 15 | 14 | ***13*** | 30 |
| 39 | 18 | ***5*** | 4 | ***3*** | 12 | 29 |
| 40 | 19 | 6 | 1 | 2 | 11 | 28 |
| 41 | 20 | ***7*** | 8 | 9 | 10 | 27 |
| 42 | 21 | 22 | 23 | 24 | 25 | 26 |
| ***43*** | 44 | 45 | 46 | 47 | 48 | 49 |

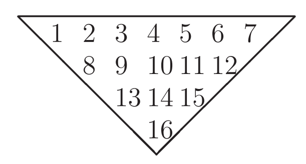
De aquí se puede notar que 8 de los 13 números posicionados en ambas diagonales son números primos, es decir, un porcentaje de $8/13 = 62\,\%$. A medida que se agregan más capas a la matriz y se verifican los números primos en las diagonales, ¿cuál es la menor longitud del cuadrado de espiral para el cual el porcentaje de primos en ambas diagonales es menor del $10\,\%$?

22. Realice un algoritmo que genere una matriz con cada uno de los recorridos que se muestran a continuación:

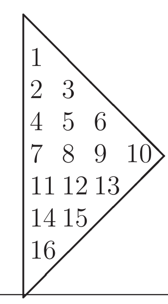
- Triángulo inferior izquierdo en forma horizontal



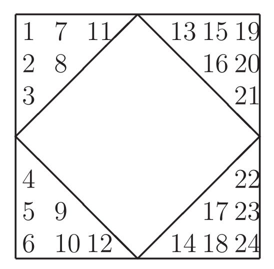

- Triángulo de Pascal invertido horizontal




- Triángulo izquierdo horizontal



- Rombo interno hueco vertical




## 4.5 Conclusiones

El desarrollo de los conceptos de vectores o arreglos en una sola dimensión (unidimensional) y en dos dimensiones (bidimensional) o matrices ha permitido:

1. Diferenciar el concepto fundamental en algoritmia de la diferencia que existe entre dirección e información; recordando que la *dirección* es el lugar de ubicación de la información dentro de la estructura de datos vectorial o matricial; en tanto que la *información* representa el contenido sintáctico y semántico de un dato que al ser almacenado o procesado en la estructura es de utilidad como resultado del esfuerzo realizado por la lógica de control del algoritmo al recorrer la estructura.

2. Identificar la forma de organización de los datos en vectores y matrices, en su diferencia fundamental; por cuanto los vectores unidimensionales son direccionados por un solo subíndice y las matrices son direccionadas por dos subíndices.

3. Conocer la utilidad de las estructuras lógicas algorítmicas al ser aplicadas en el manejo y control de las informaciones almacenadas en vectores y matrices, lo cual permite la acciones de almacenamiento, mantenimiento, ordenamiento, búsqueda, combinación, agregación y aplicación de funciones matemáticas, entre otras funciones, sobre los datos contenidos en las estructuras.

4. Aplicar la iterrelación existente entre las estructuras matemáticas con las estructuras de datos; y ambas, a su vez, interactuando con las estructuras algorítmicas. Lo cual permite soportar formalmente el almacenamiento de datos y sus lógicas de control asociadas con fundamentos matemáticos sólidos; soporte que es independiente tanto de los lenguajes de programación como de la tecnología; justificado por el hecho de que está soportado por la matemática y la algoritmia.

5. Ejemplificar el diseño de algoritmos matemáticos, de ordenamientos, de mezclas, de teoría de juegos y aplicado a sistemas universitarios y de administración de personal, con el objetivo de comprobar la utilidad de las estructuras de datos de vectores y matrices que al ser manejadas en sus direcciones por las estructuras algorítmicas permiten la resolución de problemas del mundo real de naturaleza teórica y aplicada.

6. Identificar las bases fundamentales en estructura de datos y algoritmia para el aprendizaje y análisis de estructuras de datos más complejas, tales como las teorías de listas, árboles y grafos.

7. Conocer los fundamentos iniciales de la utilización de las estructuras lógicas algorítmicas en su relación con las estructuras de datos, a fin de explorar, construir y analizar tipos de algoritmos más elaborados, tales como: los algoritmos voraces, los algoritmos recursivos, los algoritmos aplicando divide y vencerás, o la lógica algorítmica de *backtracking* o regreso hacia atrás, entre otros tipos.

# Bibliografía

Baase, S. & Van, A. (2002). *Algoritmos Computacionales. Introducción al análisis y diseño.* México: Pearson Eduction México, S. A. de C.V.

Deitel, H. & Deitel, P. (2004). *Cómo Programar en C++.* México: Pearson Eduction México, S. A. de C.V.

Tenenbaum, A. & Augestein, M. (1981). *Estructura de datos en Pascal.* Madrid: Editorial Prentice/Hall Internacional.

Tremblay, J., D. J. & Daoust, D. (1989). *Programming in Modula-2.* New York: MacGraw-Hill Book Company.

# Capítulo 5

# Funciones y procedimientos

A medida que se escriben más algoritmos o se codifican estos en un lenguaje de programación aumenta su complejidad, y posiblemente el poder realizar posteriores modificaciones se convierte en un problema. Debido a esto, los lenguajes de programación han avanzado en la forma de estructurar el código. Conviene observar la evolución y los avances para poder comprender el "¿porqué? y ¿para qué?" de un proceso de modularización o encapsulación del código o instrucciones de un programa.

En los inicios de la programación, los programadores codificaban numerosas líneas de código a partir de las estructuras de control. Las tareas de corrección y actualización se dificultaban, y por lo tanto, en ocasiones era mejor desarrollar una nueva aplicación que adecuar una existente.

Al cabo de un tiempo, los lenguajes de programación evolucionaron a lenguajes que contaban con una característica de gran apoyo, "la programación procedimental". Esta herramienta ayudó a los programadores a simplificar muchas tareas que se solían ejecutar dentro del flujo de los programas, y a emplear una cantidad menor de horas en el entendimiento y posterior modificación de este.

La programación procedimental fue un gran avance para el desarrollo de aplicaciones. Mejoró técnicamente la construcción de programas y además ayudó a entender la forma como funciona el flujo de un programa, ya que se empezó a modularizar segmentos de código que resolvían subproblemas del problema general.

De acuerdo con lo anterior, un problema complejo (Pc) requiere de su fraccionamiento en $n$ subproblemas (Spi, $1 \leq i \leq n$) tales que al dividir el problema (Pc) en subproblemas (Spi) de complejidad menor se logre la solución del problema con alto grado de complejidad. En este sentido, "Cada subproblema se resuelve mediante un módulo" (López & Vega, 2009, p. 9). Luego, al ser integrados los resultados de los subproblemas mencionados, se obtiene la solución requerida por el usuario.

Se tiene, entonces, de acuerdo con lo anterior, que la solución de un problema (Pc) en ingeniería susceptible de ser tratado computacionalmente debe tener en

cuenta las teorías de algoritmos y la de lenguajes de programación (C++, Java) para generar el software representativo del SIM. En tal sentido,

> El propósito de la ingeniería de software es permitir al diseñador enfrentar el problema de construcción de software como un problema de ingeniería, con guías y principios concretos, al igual que con herramientas de evaluación y validación. (Villalobos, 1996, p. 39)

En tal sentido, un Sistema Integrado de Manufactura (SIM) de apoyo a una empresa industrial tiene a nivel informático muchas componentes. En términos generales, el SIM está compuesto por: 1) Proyectos de producción, desarrollados por la empresa. 2) Empleados, que trabajan en los proyectos relacionados. 3) Productos, fabricados por la empresa. 4) Proveedores, que suministran las materias de primas para la construcción de los productos. 5) Inventarios, de materias primas, productos en proceso y productos terminados para almacenar el flujo de materiales de fabricación de la compañía. 6) Ventas, que es el resultado de la producción de la empresa. 7) Cuentas por pagar, que son los compromisos de la empresa para con los proveedores. 8) Cuentas por cobrar, que representa la cartera por recuperar en la empresa. 9) Contabilidad, que integra el plan contable de la empresa, y 10) Transporte, que representa el apoyo logístico de la compañía para la distribución de los productos.

Teniendo como base el SIM antes descrito, es de esperarse que cada una de las componentes genere resultados diferentes que deben ser evaluados y validados, pero que no pueden ser generados por un único algoritmo. El cumplimiento del plan de producción semanal como informe es diferente a la liquidación de la nómina de los empleados de la empresa. Luego, se habla del subsistema de producción, del subsistema de nómina y, consecuentemente, de los demás subsistemas.

Dentro del subsistema de producción se estructuran módulos que desempeñan funciones específicas, tales como: planificador de la producción y el controlador de la producción. El *planificador de la producción* es un módulo que organiza automáticamente la programación diaria, semanal o mensual de los empleados, maquinarias y materias primas asociadas a la fabricación de un determinado producto dentro de un proyecto de la empresa; por su parte, el *controlador de la producción* también es un módulo que se encarga de generar como información los indicadores de producción, tales como el indicador de productos defectuosos = (Número de productos defectuosos/Número total de productos fabricados)*100 dentro del proyecto de producción que se está desarrollando.

Cada uno de los módulos a nivel algorítmico está compuesto por estructuras lógicas básicas que desempeñan una determinada función, se identifican por un nombre correspondiente a la función que desempeñan y reciben unos parámetros de función para poder dar resultados.

Un algoritmo que apoye un procesador matemático, tal como una calculadora, se compone de varias funciones, y cada una de las funciones ejecuta su trabajo a través de un módulo. Así, si se va a operar la suma de dos números en una calculadora se requiere el Operando_1, el Operando_2 y el operador, que es el signo de la suma para dar el resultado (Operando_1 = 15, Operando_2 = 47, entonces al digitar el

15, el operando +, el número 47 y finalmente la tecla igual el resultado es 62, que es igual a la suma). Si se opera el módulo de la función factorial, representada por $Fac(n)$ con parámetro de entrada 5, al digitar en la calculadora la función $Fac$, su parámetro de entrada (5) y la tecla igual, el resultado de la función factorial es 120.

En esencia, dentro de la programación procedimental nacieron algunos conceptos, tales como funciones y subrutinas o procedimientos, que constituyen el tema de este capítulo.

# 5.1 Subrutinas

La *subrutina* es una estructura algorítmica que realiza en su lógica de control una tarea específica de entradas, procesos o salidas, cuyo flujo de control funciona a nivel autónomo. La subrutina en su lógica de control se construye con base en las estructuras lógicas expuestas en los capítulos anteriores; es decir, son propias de las subrutinas las entradas (Lea $V(i)$, como elemento de un vector $V$), procesos (Para, Mientras que, Haga_Hasta) o resultado generado por la subrutina como salida de información procesada (Escriba $M[i, j]$ variando $1 \leq i, j \leq n$, como elementos resultados de un proceso matricial $M_{n\times n}$).

Las subrutinas o procedimientos son segmentos de código que buscan simplificar tareas repetitivas dentro de un algoritmo. Se caracterizan por que no devuelven ningún valor y, además, reducen las líneas de código, de tal manera que el proceso de entendimiento sea más rápido en un proceso de modificación.

## 5.1.1 Llamado de Subrutinas

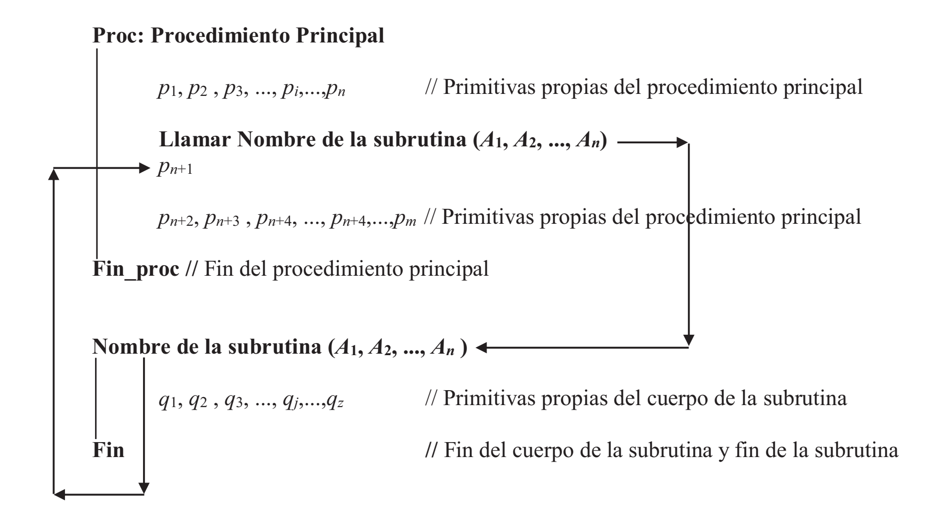
Una vez construidas las estructuras lógicas de la subrutina, el procedimiento de invocación algorítmico de la misma es como sigue:

**Llamar Nombre-de-la-Subrutina** $(A_1, A_2, \ldots, A_n)$

Este paso permite invocar la subrutina con el nombre dado. El conjunto $A$, denotado con mayúscula y cuyos elementos son $A_1$, $A_2$, ..., $A_n$, representa los parámetros del procedimiento que será llamado; estos valores se identifican también como argumentos de la subrutina. Un elemento del conjunto de argumentos ($A_i$) puede ser de entrada o salida. La decisión, si es de entrada o salida, tiene el siguiente análisis: *i*) El procedimiento principal (o programa principal) llama a la subrutina para que realice un trabajo lógico (propio o autónomo de la subrutina); luego, es lógico pensar que el programa principal debe enviar a la subrutina algunos parámetros o argumentos ($A_i$) para su cálculo; en tal sentido, dichos argumentos van del programa principal a la subrutina ($A_i \downarrow$) o de salida de la subrutina al procedimiento principal ($A_i \uparrow$). *ii*) La subrutina hace un trabajo aritmético/lógico, y sus resultados

en términos de presentación al usuario tienen dos opciones: la primera, regresar los argumentos resultados ($A_j$) ($A_i$ y $A_j$ son disyuntos) al procedimiento llamador, en cuyo caso son parámetros de salida de la subrutina ($A_j \uparrow$) y parámetros de entrada al programa principal ($A_j \downarrow$). La segunda, presentar los resultados al usuario en la subrutina; proceso en el cual la subrutina puede o no volver parámetros al procedimiento invocador.

La notación algorítmica usual para llamar la función dentro de un procedimiento principal es la siguiente:




## 5.2 Funciones

Otra cuestión fundamental en la programación procedimental es el uso de las *funciones*, que a diferencia de los procedimientos, pueden devolver un resultado después de ejecutar las instrucciones en el cuerpo de la función. Las funciones son desarrolladas para indicarle al que la invoca que se ha realizado una tarea que resuelve un pequeño problema, que arroja un resultado.

Inicialmente los lenguajes de programación mantenían unas reglas que indicaban que las funciones solo devolvían un dato, y no podían devolver un arreglo; lo cual hoy en día se mantiene a nivel de pseudocódigo, pero a nivel de programación ha evolucionado de tal manera que las funciones son capaces de devolver objetos (a nivel de programación orientada a objetos), arreglos de datos u objetos o los propios datos. Debido a esto, un arreglo de datos que se pueden arrojar por medio de una función también se pueden obtener a través de una subrutina que posea un parámetro de referencia de múltiples datos (arreglos), lo cual no era con la teoría tradicional de funciones, debido a que las funciones solo devolvían un dato.

Es importante señalar que de los dos lenguajes de programación en los que se

están incursionando, C++ maneja la teoría tradicional, mientras que Java maneja las nuevas tendencias de uso mencionadas con anterioridad.

El concepto de función matemática en su notación formal se explicita como $y = f(x)$, donde se dan valores a $x$ y se obtienen valores de $y$. Entonces el valor de $x$, o variable independiente, es el parámetro de entrada a la función, en tanto que el valor de $y$ es el resultado de la función. Concretamente, $y = f(x) = x * x$ si $x = 5$ es un argumento de entrada a la función cuadrática; su resultado en el eje de las $y$ es igual a 25.

**Identidad de las funciones**.

Las funciones que permiten la modularización de los algoritmos son definidas en notación formal algorítmica de la siguiente forma:

La semántica de la notación expuesta anteriormente es:

- Nombre de la función: Identifica la función llamada, o lo que es equivalente, es el nombre que le da identidad y autonomía algorítmica a la función, como estructura de control independiente que le permite generar sus propios resultados. Recibe también el nombre de *encabezado de la función*.
- Datos transmitidos a la función: Son los parámetros ($A_i$) que se pasan esta y que le permiten, con base en los parámetros recibidos, producir los resultados esperados mediante cálculos aritmético/lógicos.

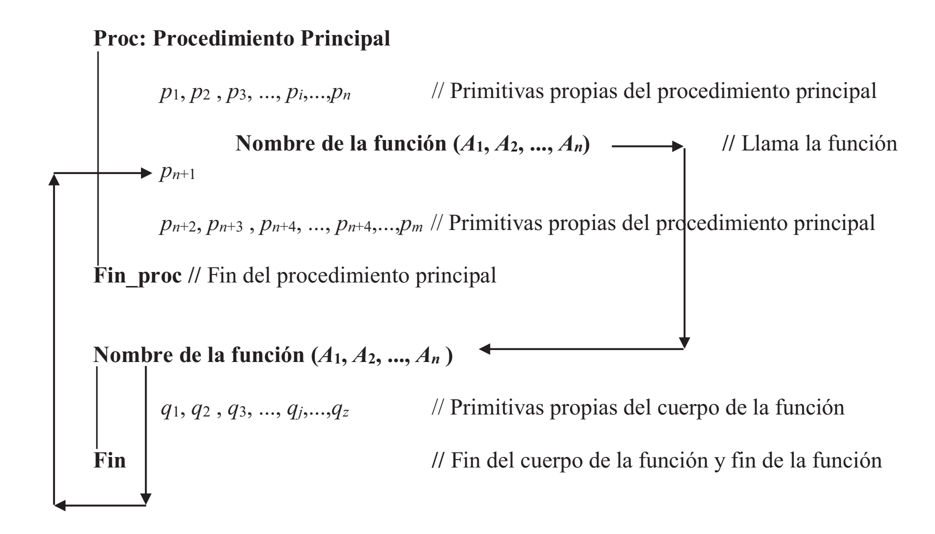
**Llamado de funciones**.

Una vez se ha definido la función, la notación de llamado algorítmico de la función desde el procedimiento o algoritmo prinicipal es

**Nombre de la función** ($A_1$, $A_2$, ..., $A_n$)

Este paso permite invocar la función con el nombre dado. De igual forma que las subrutinas, el conjunto $A_i$ mayúscula, y cuyos elementos son $A_1$, $A_2$, ..., $A_n$, representa los parámetros del procedimiento o algoritmo llamador (algoritmo principal o programa principal); parámetros que permiten transmitir datos a la función. Es de la mayor importancia resaltar que la función regresa un solo valor como resultado al procedimiento o algoritmo principal.

La notación algorítmica que se utiliza para llamar la función dentro de un procedimiento principal es la siguiente:




## 5.3 Elementos característicos

Los *procedimientos* y las *funciones* son herramientas que sirven para el proceso de formación en programación; por esto debemos saber identificar sus elementos característicos.

**Parámetros:** Son elementos opcionales, que se representan por medio de variables y/o objetos necesarios, para que se pueda resolver el subproblema para el cual se diseñó el procedimiento.

Para que las acciones descritas en una función o en un procedimiento sean ejecutadas, el programa debe llamarlos proporcionándoles los argumentos necesarios para realizar esas acciones; a estos argumentos se les conoce como **parámetros actuales**. A los parámetros definidos en la cabecera del procedimiento y de la función se los denomina **parámetros formales**.

Cuando el programa hace el llamado de una función o de un procedimiento se establece una correspondencia uno a uno entre los parámetros actuales y los formales.

En la definición del subprograma se debe especificar cómo desea que se comporte cada uno de los parámetros; esto lleva a clasificarlos así:

- **Parámetros de entrada (E):** Permiten únicamente la transmisión de información desde el programa llamador al subprograma.
- **Parámetros de salida (S):** Solo devuelven resultados.
- **Parámetros de entrada/salida (E/S):** Actúan en los dos sentidos, tanto mandando valores al subprograma como devolviendo resultados desde el subprograma al programa llamador.

### 5.3.1 Paso de parámetros

En los lenguajes de programación se manejan dos tipos de parámetros:

- **Parámetros por valor**

  Son parámetros que en el momento de ejecutarse una subrutina o función, estos valores son copiados en la memoria, de modo que si estos son modificados dentro del cuerpo de la subrutina, la variable y/o objeto que hayamos pasado con este tipo de parámetro no tendrá ningún cambio.

  **Nota:** *En las subrutinas se pueden pasar objetos, los cuales son instancias de clases, y estas hacen parte de la programación orientada a objetos; tema bien interesante que se tratará más adelante.*

- **Parámetros por referencia**

  Son parámetros que en el momento de ejecutarse una subrutina o función son referenciados por su dirección de memoria; por lo tanto, si realizamos cualquier modificación de este tipo de parámetros dentro del cuerpo de la función o subrutina, los parámetros cambiarán después de finalizar la ejecución.

  **Nota:** *Los punteros son un tema más de estudio de la programación, que vale la pena mencionar, y que hacen parte de la asignación de espacio en memoria dinámicamente.*

La forma de identificar cuándo es un parámetro de valor o de referencia en pseudocódigo no es muy legible a nivel de sintaxis; por eso es necesario que se realicen operaciones dentro del código para que se suponga que usted está manejando un parámetro de referencia.

Por ejemplo:

**sumar_numeros**($A$, $B$, $C$)
| $C \leftarrow A + B$
**Fin_sumar_numeros**

De este ejemplo se puede aclarar cómo identificar el tipo de parámetro que se usa en un procedimiento. Este posee tres parámetros: dos que se usan para poder realizar la tarea que resuelve el problema y otra para poder almacenar el resultado de la operación aritmética debido a que las subrutinas no devuelven ningún valor. También se puede decir que la variable $C$ es el único parámetro de referencia, porque solo a él se le está haciendo una asignación dentro del procedimiento.

### 5.3.2 Diferencias

La diferencia fundamental entre las funciones y subrutinas radica en que la función regresa un (1) solo valor; en tanto que la subrutina regresa varios valores. La comparación de sus diferencias se presenta en la tabla 5.1.

## 5.4 Restricciones en el nombramiento

Así como existen reglas para darles nombres a las variables en un lenguaje de programación, también se debe aclarar que las funciones y procedimientos deben poseer un nombre, que debe cumplir las siguientes reglas:

1. Los nombres de funciones o procedimientos no deben empezar por números.
2. No pueden usarse algunos caracteres especiales, tales como: ], @, —, %, ♯, $, &, /, ), (, ", !, ?, ¡, ¿, ' , etc.
3. El único caracter especial que soporta es el "_".
4. Las famosas *Reserved Words*, o palabras reservadas, son aquellas palabras que no se les pueden asociar o establecer a una variable, función o procedimiento.

**Ejemplos de nombre de funciones o procedimientos**

| | | |
|---|---|---|
| _calcular | Calculo_factorial | Techo |
| Calculo_1_factorial | Piso1 | RAIZ2 |

**Aclaracion:** *Al asignarle un nombre a una subrutina o a una función es necesario darle un nombre que sea el más adecuado de acuerdo con la tarea que vaya a realizar.*

*Tabla 5.1* Estructuras lógica de control de una función y una subrutina

| **Función** | **Subrutina** |
|---|---|
| Objetivo: Cálculo de la función factorial | Objetivo: Sumar las matrices $A$ y $B$ en la matriz $C$ ($C = A + B$) |
| **Proc**: Calcular Factorial<br>**Entero:** $n$, $r$<br>**Lea** $n$<br>$r \leftarrow$ Factorial($n$)<br>**Escriba** $r$<br>**Fin_proc**<br><br>**Factorial(**$n$**)**<br>**Entero:** $i$, $fact$<br>**Si**($n = 0$)**Entonces**<br>Devuelva 1<br>**Sino**<br>$fact \leftarrow 1$<br>**Para**($i = 1, n, 1$)**Haga**<br>$fact \leftarrow fact * i$<br>**Fin_Para**<br>**F.Si**<br>Devuelva $fact$<br>**Fin_Factorial** | **Proc**: Suma Matrices<br>**Entero:** $n$, $A[50, 50]$, $B[50, 50]$, $C[50, 50]$<br>**Lea** $n$<br>LeerMatriz($A, n$)<br>LeerMatriz($B, n$)<br>SumarMatrices($A, B, C, n$)<br>EscribirMatriz($C, n$)<br>**Fin_proc**<br><br>**LeerMatriz(**$T, n$**)**<br>**Entero:** $i$, $j$<br>**Para**($i = 1, n, 1$)**Haga**<br>**Para**($j = 1, n, 1$)**Haga**<br>**Lea** $T[i, j]$<br>**Fin_Para**<br>**Fin_Para**<br>**Fin_LeerMatriz**<br><br>**SumarMatrices(**$A \downarrow, B \downarrow, C \uparrow, n \downarrow$**)**<br>**Entero:** $i$, $j$<br>**Para**($i = 1, n, 1$)**Haga**<br>**Para**($j = 1, n, 1$)**Haga**<br>$C[i, j] \leftarrow A[i, j] + B[i, j]$<br>**Fin_Para**<br>**Fin_Para**<br>**Fin_SumarMatrices**<br><br>**EscribirMatriz(**$R, n$**)**<br>**Entero:** $i$, $j$<br>**Para**($i = 1, n, 1$)**Haga**<br>**Para**($j = 1, n, 1$)**Haga**<br>**Escriba** $R[i, j]$<br>**Fin_Para**<br>**Fin_Para**<br>**Fin_EscribirMatriz** |
| | $\downarrow$: Variable no modificada en la subrutina; $\uparrow$: Variable modificada en la subrutina |

# 5.5 Algoritmos resueltos

## 5.5.1 Funciones

Ejemplo 5.1 Contador de dígitos de un número decimal

Diseñe un algoritmo para leer un número entero $n$ y escriba una función que tome dicho número entero y devuelva el número de dígitos del número.

*Análisis*: Suponga $n = 798775$, entonces el número de dígitos del número $n$ es 6. Se van a desarrollar tres procedimientos: 1) El procedimiento que resuelve el contador sin utilizar el llamado de funciones, que se llamará (Proc1); 2) el procedimiento solución al contador requerido, utilizando el llamado a funciones y que imprime el valor resultado en la función, llamado que se identificará como procedimiento dos (Proc2); 3) el procedimiento solución al conteo requerido, pero que imprime los resultados en el procedimiento principal, el cual recibirá el nombre de procedimiento tres (Proc3).

**Proc1**: Contador decimal // Calcula el contador sin utilizar funciones
  **Entero:** $n1$, $cd$
  **Real:** $n$
  **Lea** $n$
  $n1 \leftarrow n$
  $cd \leftarrow 0$
  **Mq**($n1 > 0$)**Haga**
    $n1 \leftarrow n1/10$
    $cd \leftarrow cd + 1$
  **Fin_Mq**
  **Escriba** "El número decimal ", $n$
  **Escriba** "Tiene un número de dígitos igual a ", $cd$
**Fin_proc1**

**Proc2**: Conteo decimal // Resuelve utilizando el llamado de una función
  **Real:** $n$
  **Entero:** $temp$
  **Lea** $n$
  $temp \leftarrow$ Contador_Decimal($n$)
**Fin_proc2**

**Contador_Decimal($n$)**
  **Entero:** $n1$, $cd$
  $n1 \leftarrow n$
  $cd \leftarrow 0$
  **Mq**($n1 > 0$)**Haga**
    $n1 \leftarrow n1/10$
1 2

1 2

$cd \leftarrow cd + 1$
**Fin_Mq**
**Escriba** "El número decimal ", $n$
**Escriba** "Tiene un número de dígitos igual a ", $cd$
Devuelva 0
**Fin_Contador_Decimal**

**Proc3**: Conteo de decimales
**Entero:** *contador*
**Real:** $n$
**Lea** $n$
*contador* $\leftarrow$ Contador_Decimal($n$)
**Escriba** "El número ", $n$, " tiene ", *contador*, " cifras"
**Fin_proc3**

**Contador_Decimal($n$)**
**Entero:** $n1$, $cd$
$n1 \leftarrow n$
$cd \leftarrow 0$
**Mq**($n1 > 0$)**Haga**
$n1 \leftarrow n1/10$
$cd \leftarrow cd + 1$
**Fin_Mq**
Devuelva $cd$
**Fin_Contador_Decimal**

---

Ejemplo 5.2 Verificador de un número impar y primo

Diseñe un algoritmo compuesto por un programa principal y dos funciones, así: *i*) El programa principal, identificado como Verificador_impar_primo, lee un número $k$ que pertenece a los enteros positivos $\mathbb{Z}^+$ y verifica con una función (impar) si el número leído es impar; y en este caso, a través de la función primo verifica si el número es primo. *ii*) La estructura algorítmica de la función identificada como impar retorna al programa principal un 1 si el número es impar o un cero si el número es par. *iii*) La función primo retorna un 1 al programa principal en el caso de que el número sea primo.

*Análisis*: Sea $k = 8$ el valor de entrada, entonces el programa principal verifica que el número en primer lugar es par y, por lo tanto, no es primo. Si el valor de entrada es $k = 17$, entonces el programa ha de verificar que el número es impar (17 es impar); una vez verificado que el número es impar, entonces llama la función de chequeo de números primos (17 es primo), para dar como resultado el mensaje de "impar primo".

```
Proc: Verificador_impar_primo
  // Verifica si el número es impar con el llamado a la función Impar
  // Verifica que el número impar sea primo con el llamado a la función
  // primo
  Entero: n, rimp, rprimo
  Lea n
  // Se llama a la función Impar enviando el dato n
  // Recibe del retorno de la función 1 (si n es impar)
  rimp ← Impar(n)
  Si(rimp = 1)Entonces
    rprimo ← Primo(n)
    Si(rprimo = 1)Entonces
      Escriba "El número es impar y es primo"
    Sino
      Escriba "El número es impar y no es primo"
    F.Si
  Sino
    Escriba "El número no es impar"
  F.Si
Fin_proc

Impar(n)
  Si(n mod 2 ≠ 0)Entonces
    Devuelva 1
  Sino
    Devuelva 0
  F.Si
Fin_Impar

Primo(n)
  Entero: swp, i
  swp ← 1
  i ← 2
  Mq(swp = 1 y i ≤ √n)Haga
    Si(n mod i = 0)Entonces
      swp ← 0
    F.Si
    i ← i + 1
  Fin_Mq
  Si(swp = 1)Entonces
    Devuelva 1
  Sino
    Devuelva 0
  F.Si
Fin_Primo
```

Ejemplo 5.3 Cálculo de los cuadrados de los números enteros almacenados en un vector $V$ por sumas sucesivas

Dado un vector $V$ de $n$ elementos, el cual lee un conjunto de número enteros ($n$), diseñe: $i$) Un programa principal que lea los $n$ elementos del vector $v$ y llame una función que devuelva para cada número del vector $V$ el cálculo del cuadrado del número almacenado en el vector $V$, y lo asigne a un vector paralelo a $V$ llamado $C$. $ii$) Diseñe la función que devuelva el cuadrado de cada uno de los números, haciéndolo por sumas sucesivas; llame la función Cuadrado_Suc.

*Análisis*: Considere como supuesto que el vector $V$ tiene 10 elementos y el vector paralelo de cuadrados llamado $C$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $V$ : | 2 | 4 | 5 | 7 | 6 | 3 | 1 | 8 | 9 | 10 |
| $C$ : | 4 | 16 | 25 | 49 | 36 | 9 | 1 | 64 | 81 | 100 |

Pero los cuadrados de los números del vector V y almacenados en el vector C deben ser calculados por sumas sucesivas, así:

Para el número 7 se tiene: $7 + 7 + 7 + 7 + 7 + 7 + 7 = 49$.

**Proc**: Cálculo de los cuadrados de V
  **Entero:** $n$, $i$, $V[50]$, $C[50]$
  **Lea** $n$
  $i \leftarrow n$
  **Para**$(i = 1, n, 1)$**Haga**
    **Lea** $V[i]$
  **Fin_Para**
  // Llamado de la función que calcula los cuadrados de los números
  **Para**$(i = 1, n, 1)$**Haga**
    $C[i] \leftarrow$ Cuadrado_Suc$(V[i])$
  **Fin_Para**
  // Impresión de los cuadrados sucesivos contenidos en el vector $C$
  $i \leftarrow 1$
  **Mq**$(i \leq n)$**Haga**
    **Escriba** "El número contenido en la posición ", $i$, " del vector "
    **Escriba** $V[i]$, " tiene cuadrado ", $C[i]$
    $i \leftarrow i + 1$
  **Fin_Mq**
**Fin_proc**

**Cuadrado_Suc**($m$)
  **Entero:** $r$, $j$
  $r \leftarrow 0$
  $j \leftarrow m$
  **HH**
    $r \leftarrow r + m$
    $j \leftarrow j - 1$
  **Fin_HH**($j = 0$)
  Devuelva $r$
**Fin_Cuadrado_Suc**

---

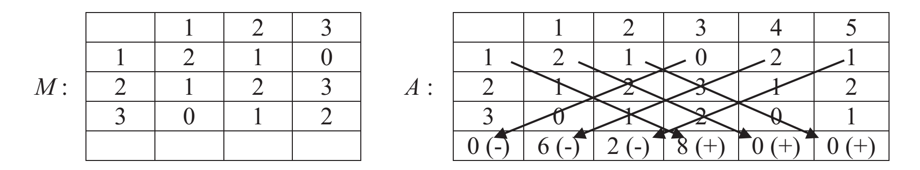
Ejemplo 5.4 Cálculo del Máximo Común Divisor desde un programa principal

Diseñe un algoritmo compuesto por: $i$) Un procedimiento principal identificado como Principal_MCD, el cual lee dos números, $n$ y $m$, verificando que pertenecen a los enteros positivos ($\mathbb{Z}^+$); llame a una función identificada como MCD, la cual devuelve el Máximo Común Divisor entre $n$ y $m$; e imprima el MCD en el programa principal . $ii$) Diseñe la función que calcule el Máximo Común Divisor (MCD) de los dos números $(n, m)$.

*Análisis*: Suponga que los número leídos son $n = 12$ y $m = 3$, entonces el concepto de MCD son los factores comunes de ambos números con el menor exponente. Luego, los factores de 12 son $2^2$, 3 y 1 (porque $2^2 * 3 * 1 = 12$) y los factores del número 3 son el 3 y 1 (porque $3 * 1 = 3$); por lo tanto, el factor común es el 3; en este caso con su menor exponente, que es 1. Para el cálculo del máximo común divisor se utilizará el algoritmo de Euclides, es decir, $(a, b) = (a - b, b)$. Por ejemplo,

$$(24, 16) = (24 - 16, 16) = (8, 16) = (8, 16 - 8) = (8, 8) = 8.$$

**Proc**: Máximo Común Divisor
  // Lógica de control del programa principal del MCD
  **Entero:** $n$, $m$, $r$
  **Lea** $n, m$
  **Si**($n > 0$ y $m > 0$)**Entonces**
    $r \leftarrow$ MCD($n, m$)
    **Escriba** "Los números ", $n$, $m$
    **Escriba** "Tienen como máximo común divisor el valor de ", $r$
  **Sino**
    **Escriba** "Los números no pertenecen a los enteros positivos"
  **F.Si**
**Fin_proc**



M:

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 2 | 1 | 0 |
| 2 | 1 | 2 | 3 |
| 3 | 0 | 1 | 2 |
| | | | |

A:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 2 | 1 | 0 | 2 | 1 |
| 2 | 1 | 2 | 3 | 1 | 2 |
| 3 | 0 | 1 | 2 | 0 | 1 |
| 0 (-) | 6 (-) | 2 (-) | 8 (+) | 0 (+) | 0 (+) |

*Figura 5.1* Cálculo del determinante de una matriz por el método de Sarrus

**MCD**$(u, v)$
| **Entero:** $m$
| **Mq**$(u > 0)$**Haga**
| | **Si**$(u < v)$**Entonces**
| | | // Se intercambian $u$ y $v$
| | | $m \leftarrow u$
| | | $u \leftarrow v$
| | | $v \leftarrow m$
| | **F.Si**
| | $u \leftarrow u - v$
| **Fin_Mq**
| Devuelva $v$
**Fin_MCD**

---

Ejemplo 5.5 Cálculo del determinante de una matriz de orden 3 por el método de Sarrus

---

Diseñe un programa principal que : *i*) Lea una matriz de 3 líneas por 3 columnas, y llame a una función que calcule el determinante de la matriz e imprima su valor. *ii*) Diseñe una función llamada Sarrus que calcule el determinante de la matriz de orden 3.

*Análisis*: Suponga la matriz dada $M$; esta se amplía en una segunda matriz $A$, a la cual se le adicionan al final las dos primeras columnas de la matriz $M$. Posteriormente se multiplica las diagonales de la matriz ampliada $A$ y las que están a la derecha son positivas en la multiplicación y las que están a la izquierda son negativas, como se presenta en la figura 5.1.

Luego el determinante de la matriz es

$$D = 2 \cdot 2 \cdot 2 + 1 \cdot 3 \cdot 0 + 0 \cdot 1 \cdot 1 - (0 \cdot 2 \cdot 0 + \cdot 3 \cdot 1 + 1 \cdot 1 \cdot 2) = 8 - 8 = 0.$$

**Proc**: Determinante Matriz Orden 3
| // Lee la matriz de orden 3
| **Entero:** $i$, $j$
1

```
1
    Real: M[3,3], det
    Para(i = 1,3,1)Haga
        Para(j = 1,3,1)Haga
            Lea M[i,j]
        Fin_Para
    Fin_Para
    det ← Sarrus(M)
    Escriba "El determinante de la matriz es igual a: ", det
Fin_proc

Sarrus(M)
    Entero: i, j, t, dc
    Real: A[3,5], d, sumad, z, sumai, det
    Para(i = 1,3,1)Haga
        Para(j = 1,3,1)Haga
            A[i,j] ← M[i,j]
        Fin_Para
    Fin_Para
    // Columnas adicionales de la matriz ampliada
    Para(i = 1,3,1)Haga
        Para(j = 4,5,1)Haga
            A[i,j] ← A[i,j − 3]
        Fin_Para
    Fin_Para
    // Cálculo del determinante
    i ← 1
    j ← 1
    dc ← 1
    t ← 3
    d ← 1
    sumad ← 0
    Mq(i ≤ 3)Haga
        Mq(j ≤ t y t ≤ 5)Haga
            d ← d * A[i,j]
            i ← i + 1
            j ← j + 1
        Fin_Mq
        sumad ← sumad + d
        d ← 1
        i ← 1
        dc ← dc + 1
        j ← dc
        t ← t + 1
    Fin_Mq
1
```

1

$i \leftarrow 1$, $j \leftarrow 3$, $dc \leftarrow 3$
$t \leftarrow 1$
$z \leftarrow 1$
$sumai \leftarrow 0$
**Mq**($i \leq 3$)**Haga**
  **Mq**($j \geq t$ y $t \leq 3$)**Haga**
    $z \leftarrow z * A[i, j]$
    $i \leftarrow i + 1$
    $j \leftarrow j - 1$
  **Fin_Mq**
  $sumai \leftarrow sumai + z$
  $z \leftarrow 1$
  $i \leftarrow 1$
  $dc \leftarrow dc + 1$
  $j \leftarrow dc$
  $t \leftarrow t + 1$
**Fin_Mq**
$det \leftarrow sumad - sumai$
Devuelva $det$
**Fin_Sarrus**

---

Ejemplo 5.6 Chequeo de disponibilidad de anaqueles de góndolas

Una empresa comercial presenta sus productos para la venta en $k$ góndolas, cada una de las cuales tiene $n$ anaqueles, los cuales se utilizan para exponer los productos al público. Diseñe $i$) Un programa principal identificado Principal_Chequeo, que llame a una función que devuelva si un anaquel específico dado $q$, que pertenece a una góndola g, está vacío para ser utilizado. $ii$) Diseñe una función llamada Func_Disp, que devuelva 1 al programa principal si $q$, que pertenece a $g$, está disponible y 0 en caso contrario. Asuma que las $k$ góndolas y $n$ anaqueles por góndola en su disponibilidad son almacenados en una matriz de disponibilidad llamada Disp y su disponibilidad debe ser leída en el programa principal.

*Análisis*: Suponga que la empresa tiene 10 góndolas, cada una de ellas con 10 anaqueles. Adicionalmente, suponga que se necesita conocer si el anaquel numerado 10, que pertenece a la góndola 7, está ocupado.

En la figura 5.2 se puede observar que la góndola ubicada en la fila 7 en el cruce con la columna 10 que representa el anaquel tiene el valor de cero; luego el anaquel está disponible para mostrar los productos de venta de la empresa.

| Anaqueles→ Góndolas ↓ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | **10 q** |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 3 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 5 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 6 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| **7 g** | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | **0** |
| 8 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 9 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 10 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

*Figura 5.2* Chequeo matricial de disponibilidad de anaqueles en góndolas

```
Proc: Chequeo
| Entero: k, n, i, j, Disp[50,50], q, g, rd
| Lea k, n
| i ← 1
| j ← 1
| Mq(i ≤ k)Haga
| | Mq(j ≤ n)Haga
| | | Lea Disp[i, j]
| | | j ← j + 1
| | Fin_Mq
| | i ← i + 1
| Fin_Mq
| // Se hace el chequeo de la disponibilidad
| // del anaquel q en la góndola g
| Lea q, g
| rd ← FunDisp(Disp, g, q)
| Si(rd = 0)Entonces
| | Escriba "El anaquel ", q, " en la góndola ", g, " está disponible"
| Sino
| | Escriba "El anaquel ", q, " en la góndola ", g, " no está disponible"
| F.Si
Fin_proc

FunDis(Disp, g, q)
| Si(Disp[g, q] = 0)Entonces
| | Devuelva 0
| Sino
| | Devuelva 1
| F.Si
Fin_FunDis
```

### 5.5.2 Subrutinas

Ejemplo 5.7 Ordenador vectorial

Construya un algoritmo compuesto por un procedimiento principal (llamador) y tres subrutinas. El procedimiento principal lee el número de elementos en la variable $k$ del vector $V$, llama en primer lugar a la subrutina que lee los elementos del vector, en segundo lugar llama a la subrutina que ordena el vector $V$ sobre un vector temporal $T$ con el fin de conservar el vector original, finalmente el llamador invoca una tercera subrutina en la que se imprimen tanto el vector original como el vector ordenado.

*Análisis*: La arquitectura de control del ordenador vectorial se presenta en la figura 5.3, suponiendo que el vector $V$ tiene $k = 6$ elementos de información.

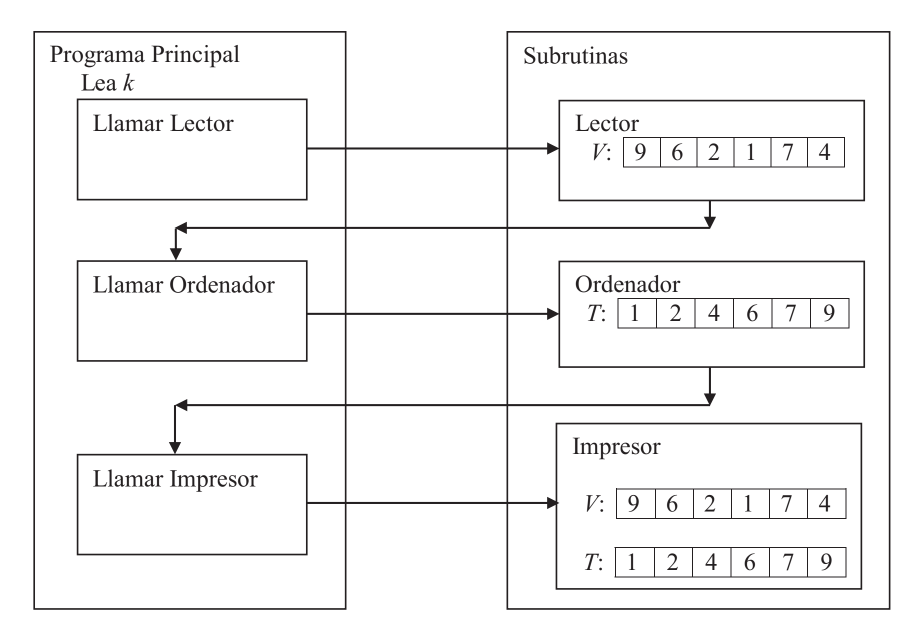



*Figura 5.3* Ordenador vectorial

**Proc**: Ordenador vectorial
  **Entero:** $k$, $V[50]$, $T[50]$
  **Lea** $k$
  Lector$(V, k)$
  Ordenador$(V, T, k)$
  Impresor$(V, T, k)$
**Fin_proc**

```
Lector(A, k)
| Entero: i
| i ← 1
| Mq(i ≤ k)Haga
| | Lea A[i]
| | i ← i + 1
| Fin_Mq
Fin_Lector

Ordenador(A, T, k)
| Entero: i, j, temp
| j ← 1
| Mq(j ≤ k)Haga
| | T[j] ← A[j]
| | j ← j + 1
| Fin_Mq
| Para(i = 2, k, 1)Haga
| | temp ← T[i]
| | T[0] ← temp
| | j ← i
| | Mq(temp < T[j − 1])Haga
| | | T[j] ← T[j − 1]
| | | j ← j − 1
| | Fin_Mq
| | T[j] ← temp
| Fin_Para
Fin_Ordenador

Impresor(V, T, k)
| Entero: i
| Escriba "Vector original V: "
| i ← 1
| Para(i = 1, k, 1)Haga
| | Escriba V[i]
| Fin_Mq
| Escriba "Vector ordenado T: "
| i ← 1
| Mq(i ≤ k)Haga
| | Escriba T[i]
| | i ← i + 1
| Fin_Mq
Fin_Impresor
```

Ejemplo 5.8 Determinante de una matriz diagonal

Diseñe un algoritmo compuesto por: *i*) Un procedimiento principal, cuyo nombre es Determinante_Matriz_Escalar, en el que se efectúe la lectura de una matriz $M$ de $n \times n$ elementos. *ii*) El llamado a una función identificada como Escalar, la cual compruebe si la matriz $M$ es escalar, y devuelva un *switch* llamado *Sw_me* en el valor de uno al procedimiento principal. *iii*) El cálculo del determinante de la matriz $M$, a través de una función identificada con el nombre de Determinante, la cual calcula el determinante de la matriz escalar.

*Análisis*: Una *matriz escalar* es aquella cuyos elementos situados en las posiciones triangular superior e inferior de la matriz son cero. El determinante de una matriz escalar se calcula multiplicando los elementos de la diagonal principal, según se muestra a continuación.

Si $M$ es una matriz de la forma

$$M = \begin{bmatrix} m_{11} & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & m_{22} & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & m_{33} & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & m_{ii} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & \cdots & m_{nn} \end{bmatrix}$$

entonces el determinante $D$ de la matriz diagonal $M$ se calcula con la expresión matemática

$$D = \det(M) = \prod_{i=1}^{n} m_{ii}$$

**Proc**: Determinante Matriz Diagonal
| **Entero:** $n$, $i$, $j$, $Sw_me$
| **Real:** $M[50, 50]$, $d$
| **Lea** $n$
| $i \leftarrow 1$
| **Mq**$(i \leq n)$**Haga**
| | $j \leftarrow 1$
| | **Mq**$(j \leq n)$**Haga**
| | | **Lea** $M[i, j]$
| | | $j \leftarrow j + 1$
| | **Fin_Mq**
1 2

```
1 2
| | i ← i + 1
| Fin_Mq
| Sw_me ← 1
| Escalar(M, Sw_me, n)
| Si(Sw_me = 1)Entonces
| | Determinante(M, d, n)
| | Escriba "El valor del determinante de la matriz es ", d
| Sino
| | Escriba "La matriz no es diagonal"
| F.Si
Fin_proc

Escalar(M, Sw_me, n)
| Entero: i, j
| i ← 1
| Mq(i ≤ n)Haga
| | j ← 1
| | Mq(j ≤ n)Haga
| | | Si(i ≠ j)Entonces
| | | | Si(M[i, j] ≠ 0 o M[j, i] ≠ 0)Entonces
| | | | | Sw_me ← 0
| | | | | j ← n + 1
| | | | | i ← n + 1
| | | | F.Si
| | | F.Si
| | | j ← j + 1
| | Fin_Mq
| | i ← i + 1
| Fin_Mq
Fin_Escalar

Determinante(M, d, n)
| Entero: i
| d ← 1
| i ← 1
| Mq(i ≤ n)Haga
| | d ← d * M[i, i]
| | i ← i + 1
| Fin_Mq
| Devuelva d
Fin_Determinante
```

Ejemplo 5.9 Productoria de matrices en cadena

Diseñe un algoritmo compuesto por un procedimiento principal que calcule la expresión matricial representativa de la productoria de $k$ matrices ($M$) cuadradas de dimensión $n$:

$$R_{n\times n} = \prod_{i=1}^{k} M_{i,\,[n\times n]}$$

El procedimiento principal debe: $i$) Leer el controlador del número de matrices que se debe multiplicar ($k$). $ii$) Llamar a una subrutina que multiplique las matrices. iii) Llamar a una subrutina que imprima la matriz resultado ($R$) de la productoria de las $M$ matrices en cadena.

*Análisis*: 1) Leer el controlador de las matrices que se debe multiplicar en la variable $k$. 2) Cargar la matriz resultado ($R$) con la primera matriz leída (llamando la subrutina Lector). 3) Leer la segunda matriz ($M$) (llamando nuevamente la subrutina Lector). 4) Llamar la subrutina Productor, la cual multiplica la última matriz leída ($M$) con la matriz previa ($R$). 5) Asignar el resultado de la multiplicación matricial ($R \times M$) que se calcula en la matriz $C$ a la matriz que acumula el resultado de la productoria ($R$). 6) Finalmente, leer la siguiente matriz y llamar nuevamente a la subrutina Productor $k - 1$ veces.

```
Proc: Productoria de Matrices en Cadena
|  Entero: k, n, i, R[50,50], M[50,50], C[50,50]
|  Lea k
|  Lea n
|  Lector(R, n)
|  i ← 1
|  Mq(i ≤ k − 1)Haga
|  |  Lector(M, n)
|  |  Productor(R, M, C, n)
|  |  Asignador(R, C, n)
|  |  i ← i + 1
|  Fin_Mq
|  Impresor(R, n)
Fin_proc

Lector(T, n)
|  Entero: i, j
|  Para(i = 1, n, 1)Haga
|  |  Para(j = 1, n, 1)Haga
|  |  |  Lea T[i, j]
|  |  Fin_Para
|  Fin_Para
Fin_Lector
```

```
Productor(A, B, C, n)
  Entero: i, j, k
  Para(i = 1, n, 1)Haga
    Para(j = 1, n, 1)Haga
      C[i, j] ← 0
    Fin_Para
  Fin_Para
  Para(i = 1, n, 1)Haga
    Para(j = 1, n, 1)Haga
      Para(k = 1, n, 1)Haga
        C[i, j] ← C[i, j] + A[i, k] * B[k, j]
      Fin_Para
    Fin_Para
  Fin_Para
Fin_Productor
Asignador(R, C, n)
  Entero: i, j
  Para(i = 1, n, 1)Haga
    Para(j = 1, n, 1)Haga
      R[i, j] ← C[i, j]
    Fin_Para
  Fin_Para
Fin_Asignador
Impresor(R, n)
  Entero: i, j
  Para(i = 1, n, 1)Haga
    Para(j = 1, n, 1)Haga
      Escriba R[i, j]
    Fin_Para
  Fin_Para
Fin_Impresor
```

Ejemplo 5.10 Contador digital

Dado un vector $V$ de $m$ dígitos, el cual contiene únicamente valores binarios (0,1), diseñe un procedimiento principal, el cual: $i$) Lee el número de dígitos binarios del vector $V$, o sea, $m$, y lee los valores de los elementos del vector. $ii$) Llama a una subrutina identificada como Contador, la cual recibe el vector $V$ en binario y calcula como resultado un vector $C$, el cual contiene el contador binario del vector $V$. $iii$) Llama a una subrutina identificada como Impresor, la cual escribe tanto el vector original ($V$) como el vector resultado del contador ($C$).

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $V$ : | 0 | 1 | 0 | 0 |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $C$ : | 0 | 1 | 0 | 1 |

*Análisis*: Suponga un número decimal $n$ igual 4, el cual en cuartetos de dígitos ($m = 4$) se expresa como $(0100)_2$; entonces la solución del contador binario implica que después del número decimal 4 sigue el 5; y el $(5)_{10} = (0101)_2$. Luego, con base en lo anterior, los valores del número 4 se almacenan en el vector de entrada $V$, y la subrutina tiene que devolver al programa principal el vector $C$, el cual contiene el contador (siguiente número) a partir del contenido del vector $V$.

```
Proc: Principal
| Entero: m, i, V[50], C[50]
| Lea m
| Para(i = 1, m, 1)Haga
| | Lea V[i]
| Fin_Para
| Contador(V, C, m)
| Impresor(V, C, m)
Fin_proc

Contador(V, C, m)
| Entero: j
| j ← 1
| Mq(j ≤ m)Haga
| | C[j] ← V[j]
| | j ← j + 1
| Fin_Mq
| j ← m + 1
| HH
| | j ← j − 1
| | C[j] ← 1 − C[j]
| Fin_HH(C[j] = 1 o j = 1)
Fin_Contador

Impresor(V, C, m)
| Entero: j
| Escriba "Los valores del vector base para el contador son: "
| j ← 1
| Mq(j ≤ m)Haga
| | Escriba V[j]
| | j ← j + 1
| Fin_Mq
| Escriba "Los valores binarios del contenido del contador son: "
| j ← 1
| Mq(j ≤ m)Haga
| | Escriba C[j]
| | j ← j + 1
| Fin_Mq
Fin_Impresor
```

Ejemplo 5.11 Gestor de Ventas

Una compañía multinacional de distribución de productos tiene a cargo $n$ vendedores, cada uno de ellos atendiendo $k$ zonas de ventas. Las zonas son asignadas por año a cada vendedor en la matriz de relación cruzada *vendedor por zona*, identificada como $A$. La empresa cada día del año, durante los 365 días, registra las ventas de cada vendedor en las zonas asignadas por vendedor. Diseñe un algoritmo para realizar la gestión de ventas de la empresa, compuesto por un procedimiento principal y conjunto de subrutinas. El procedimiento principal lee los $n$ vendedores, y debe llamar a una subrutina (Activar) que lee los códigos de los $n$ vendedores y sus zonas asignadas, y a su vez construye la matriz de activación por zona de cada vendedor durante el año; llamar a una subrutina (Relacionar) que arma una tabla relación cruzada entre vendedores_zonas; llamar a una subrutina (Cargar_Ventas) que lea una matriz de ventas, la cual contiene las ventas diarias por cada vendedor en la zona asignada; finalmente, llamar a una subrutina (Gestor) para responder las preguntas $i$) Total de ventas anuales por zona asociada al vendedor de la compañía; $ii$) Total de ventas por empleado durante el año; $ii$) Vendedor con el mayor número de zonas activadas.

*Análisis*: Las estructuras de vectores y matrices del gestor se presentan en la figura 5.4.

| TC |
|---|
| C |
| 10 |
| . |
| . |
| 20 |
| . |
| . |
| $n$ |

| Zonas ($A$) | | | |
|---|---|---|---|
| $Z_1$ | $Z_2$ | ... | $Z_k$ |
| 1 | 1 | | |
| | | | |
| | | | |
| | | | 1 |
| | | | |
| | | | |
| | | | |

| Relación | |
|---|---|
| $C_i$ | $Z_i$ |
| 10 | 1 |
| 10 | 2 |
| | |
| 20 | k |
| | |
| | |
| | |

| Venta diarias ($V$) | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | ... | 365 |
| 1 | 4 | 1 | | 1 |
| 1 | 1 | 2 | | 3 |
| | | | | |
| 2 | 1 | 2 | | 2 |
| | | | | |
| | | | | |
| | | | | |

| TVZ | TVE | CZ |
|---|---|---|
| | | |
| 7 | 14 | 2 |
| 7 | | |
| | | |
| 7 | 7 | 1 |
| | | |
| | | |
| | | |

*Figura 5.4* Vectores y Matrices del gestor de ventas

El vector de códigos de empleados ($TC$) es un vector columna o arreglo unidimensional, el cual contiene los códigos ($C$) de los $n$ empleados de la empresa. La matriz de zonas ($A$) contiene las zonas disyuntas asignadas a cada empleado, y en la celda $(i, j)$ hay un bit en 1, si el empleado $i$ es asignado a la zona $j$ $(1 \leq j \leq k)$. La matriz de relación entre empleados_zonas ($R$) es una matriz de dos columnas: en la primera se almacena el código del empleado y la segunda contiene la zona asignada al empleado con tantas líneas ($m$) como relaciones existan entre empleados y zonas; en este sentido, $R$ se conoce también como un "arrays de punteros" (Lipschutz, 1987,

p. 128) o arreglo de punteros. La matriz de ventas diarias ($V$) contiene para los 365 días del año las ventas asociadas a la $i$-ésima fila contenida en la relación $R[i, j]$. Finalmente, los vectores columna de total de ventas por zona ($TVZ$), total de ventas por empleado ($TVE$) y el contador de zona ($CZ$) almacenan los totales de ventas por zona, empleados y el contador de empleados con el mayor número de zonas activadas.

```
Proc: Gestor_de_ventas
| Entero: n, k, A[50, 50], R[50, 50, V[50, 50], TC[50], TVE[50], m
| Lea n
| Lea k
| Activar(A, n, k, TC)
| Relacionar(A, R, n, k, TC)
| Cargar_Ventas(V, m)
| Gestor(TC, A, R, V, TVZ, TVE, n, m)
Fin_proc

Activar(A, n, k, TC)
| Entero: i, Codigo, Zona
| Para(i = 1, n, 1)Haga
| | Lea Codigo
| | TC[i] ← Codigo
| | Lea Zona
| | Mq(Zona ≠ 0)Haga
| | | Si(Zona ≥ 1 o Zona ≤ k)Entonces
| | | | A[i, Zona] ← 1
| | | Sino
| | | | Escriba "Zona no válida"
| | | F.Si
| | Fin_Mq
| Fin_Para
Fin_Activar

Relacionar(A, R, n, k, TC)
| Entero: m, i, j
| m ← 1
| Para(i = 1, n, 1)Haga
| | Para(j = 1, k, 1)Haga
| | | Si(A[i, j] = 1)Entonces
| | | | R[m, 1] ← TC[i]
| | | | R[m, 2] ← j
| | | F.Si
| | | m ← m + 1
| | Fin_Para
| Fin_Para
Fin_Relacionar
```

```
Cargar_Ventas(V, m)
| Entero: i, j
| Para(i = 1, m, 1)Haga
| | Para(j = 1, 365, 1)Haga
| | | Lea V[i, j]
| | Fin_Para
| Fin_Para
Fin_Cargar_Ventas

Gestor(TC, A, R, V, TVZ, TVE, n, m)
| Entero: i, j, CZ[50]
| // Total de ventas anuales por zona de la compañía (TVZ)
| i ← 1
| Mq(i ≤ m)Haga
| | TVZ[i] ← 0
| | i ← i + 1
| Fin_Mq
| i ← 1
| Para(i = 1, m, 1)Haga
| | j ← 1
| | Mq(j ≤ 365)Haga
| | | TVZ[i] ← TVZ[i] + V[i, j]
| | | j ← j + 1
| | Fin_Mq
| Fin_Para
|
| // Total de ventas por empleado durante el año (TVE)
| Para(i = 1, n, 1)Haga
| | TVE[i] ← 0
| Fin_Para
| i ← 1
| j ← 1
| Mq(i ≤ m)Haga
| | Mq(TC[i] = R[j, 1])Haga
| | | Para(d = 1, 365, 1)Haga
| | | | TVE[i] ← TVE[i] + V[j, d]
| | | Fin_Para
| | | j ← j + 1
| | Fin_Mq
| | j ← j + 1
| | i ← i + 1
| Fin_Mq
| // Vendedor con el mayor número zonas activadas
1
```

```
1
  Para(i = 1, n, 1)Haga
  | CZ[i] ← 0
  Fin_Para
  i ← i + 1
  Mq(i ≤ n)Haga
  | j ← 1
  | Mq(j ≤ k)Haga
  | | Si(A[i, j] = 1)Entonces
  | | | CZ[i] ← CZ[i] + 1
  | | F.Si
  | | j ← j + 1
  | Fin_Mq
  | i ← i + 1
  Fin_Mq
Fin_Gestor
```

## 5.5.3 Mixtos

Ejemplo 5.12 Determinar el vector con más elementos

Dados dos vectores de tamaño $n$ y $m$, respectivamente, los cuales contienen números enteros, realice un algoritmo que determine el vector que contiene más elementos que sean mayores o iguales a su respectivo tamaño.

*Análisis*: Si el algoritmo recibe $A = [7, 2, 5, 9, 8]$, $B = [5, 9, 3, 7]$, entonces su salida debe ser '$A$ tiene más elementos mayores o iguales a su tamaño'. La función contarMayor será la encargada de contar cuántos elementos del vector son mayores o iguales al tamaño; para esto recibe como parámetros el vector y su tamaño, se declara una variable local que funcionará como un índice, $k$, que irá recorriendo cada posición del vector en un Para, donde se comparará el elemento de esa posición con el tamaño; en caso de ser mayor o igual que este, la variable que sirve como contador, $cont$, incrementará su valor en uno; al final la función devolverá el valor de $cont$.

Para la lectura de los vectores se utiliza la función leer_Vector explicada; se realiza dos veces su llamado, cada una correspondiente a un vector, mandando a cada uno con su respectivo tamaño.

En el algoritmo principal se usan 6 variables, dos vectores, $a[m], b[n]$, con sus respectivos tamaños, $m, n$, y dos variables, $CA, CB$, donde se guardará el elemento mayor correspondiente a cada vector. Lo primero que hace el algoritmo es leer los dos vectores, por medio del llamado a la subrutina leer_Vector, luego se llama a la función contarMayor enviando como parámetros el tamaño y el correspondiente vector, y el resultado se asigna a

la variable indicada; por último se comparan y se imprime qué vector tiene más elementos mayores o iguales a su tamaño.

**Proc**: Vector con más elementos
  **Entero:** $n$, $m$, $a[50]$, $b[50]$, $CA$, $CB$
  **Lea** $n, m$
  leer_Vector$(m, a)$
  leer_Vector$(n, b)$
  $CA \leftarrow$ contarMayor$(m, a)$
  $CB \leftarrow$ contarMayor$(n, b)$
  **Si**$(CA > CB)$**Entonces**
    **Escriba** "El primer vector tiene más elementos"
    **Escriba** "mayores o iguales a su tamaño que el segundo"
  **Sino**
    **Si**$(CB > CA)$**Entonces**
      **Escriba** "El segundo vector tiene más elementos"
      **Escriba** "mayores o iguales a su tamaño que el primero"
    **Sino**
      **Escriba** "Los dos vectores tienen más elementos"
      **Escriba** "mayores o iguales a su tamaño"
    **F.Si**
  **F.Si**
**Fin_proc**

**leer_Vector(**$tam$**,** $v$**)**
  **Entero:** $i$
  **Para**$(i = 1, tam, 1)$**Haga**
    **Escriba** "Digite el elemento de la posición ", $i$
    **Lea** $v[i]$
  **Fin_Para**
**Fin_leer_vector**

**contarMayor(**$tam$**,** $v$**)**
  **Entero:** $cont$, $k$
  $cont \leftarrow 0$
  **Para**$(k = 1, tam, 1)$**Haga**
    **Si**$(v[k] >= tam)$**Entonces**
      $cont \leftarrow cont + 1$
    **F.Si**
  **Fin_Para**
  $contarMayor \leftarrow cont$
**Fin_leer_vector**

---

Ejemplo 5.13 Redondeo

Implemente una función que devuelva el número real positivo $a$ con $b \geq 1$ cifras.

*Análisis*: Antes de continuar se analizarán posibles entradas. Por ejemplo, si $a = 1.467$ y $b = 2$, entonces la salida debe ser 1.4. Una forma de realizar esto es contando cuántas cifras tiene $a$ y eliminando las últimas cifras, de tal forma que el número solamente quede con $b$ cifras. Para contar cuántas cifras tiene $a$ se tiene que llevar a notación científica, puesto que ceros a la izquierda o ceros a la derecha no cuentan. Por ejemplo, el número 0.00123 tiene en realidad 3 cifras. El número 12000 tiene en realidad 2 cifras. Sin embargo, el número 10100 tiene tres cifras.

El primer paso consiste en llevar el número a notación científica; para ello hay que llevar el número real entre 1 y 10. Por ejemplo, si el usuario ingresa 0.00123, entonces este número debe ser llevado a 1.23, y a su vez se debe contar cuántas veces se multiplicó o dividió por 10. Si el usuario ingresa un número real $a$, la primera parte de la función volverá este número $a = ca \times 10^{cont}$. Donde $ca$ es un número real entre 1 y 10 y $ca$ es un entero.

Una vez se tiene $ca$, ya es posible iniciar el proceso de truncado. Este número se multiplica por $10^{b-1}$, de tal forma que la parte entera tendrá $b$ cifras (recuerde que ya tiene 1) y la parte fraccionaria tendrá el resto de cifras. Por ejemplo, si $ca = 1.232345$ y $b = 3$, entonces $ca$ se multiplica por $10^{3-1} = 10^2$. Es decir,

$$ca \times 10^{3-1} = ca \times 10^2 = 123.2345.$$

Observe que la parte entera es 123 y tiene exactamente el número de cifras deseadas. El número $ca \times 10^2$ se redondea y luego se divide por $10^2$, de manera que se obtiene el número $a$ con $b$ cifras significativas. La función redondear redondea un real al entero más cercano.

**Entrada:** $a = 1.467, b = 2$
**Salida:** El número redondeado es 1.47.

```
redondea(a, b)
| Entero: cont
| Real: ca
| cont ← 0
| ca ← a
| Si(ca > 10)Entonces
| | Mq(ca > 10)Haga
| | | ca ← ca/10
| | | cont ← cont + 1
| | Fin_Mq
| Sino
| | Si(ca < 1)Entonces
| | |
1 2 3
```

```
1 2 3
| | | Mq(ca < 1)Haga
| | | | cont ← cont − 1
| | | Fin_Mq
| | F.Si
| F.Si
| ca ← redondear(ca * 10^(b−1))
| ca ← ca/10^(b−1)
| Devuelva ca * 10^cont
Fin_redondea
```

---

Ejemplo 5.14 Mayor elemento, transpuesta y otros

---

Sea una matriz $A$ de $N \times M$, halle: $a)$ El mayor elemento de la matriz, $b)$ su transpuesta y $c)$ un vector con todos los elementos impares de la matriz.

**Entrada:**

| | | |
|---|---|---|
| 15 | 62 | 30 |
| 8 | 45 | 80 |
| 28 | 38 | 38 |

**Salida:**

1. El mayor elemento de la matriz es 80.
2. 

| | | |
|---|---|---|
| 15 | 8 | 28 |
| 62 | 45 | 38 |
| 30 | 80 | 38 |

3. Los números impares en la matriz son $A = [15, 45]$.

*Análisis*: Como su nombre lo indica, leer_Matriz es la subrutina encargada de leer los números que se guardarán en la matriz. Recibe como parámetros una matriz, $M$, su número de filas, $fila$, y de columnas, *columna*. Para recorrer la matriz se utilizan dos ciclos Para: el primero para acceder a las filas y el segundo para las columnas utilizando los índices $i, j$, respectivamente. Para cada posición $[i, j]$ se pedirá digitar un número, el cual será guardado en la matriz en ese lugar.

La subrutina mayor recibe como parámetros una matriz, su número de filas y de columnas. Se declaran tres variables enteras: $i, j$, que se emplean como índices, y $May$, donde se guardará el mayor elemento de la matriz. Para hacer el recorrido se utilizan dos ciclos Para: el más externo recorre las filas y el interno las columnas; cuando el algoritmo inicia se considera como número mayor el cero (0), pero cada vez que se encuentra uno mayor que el que se tiene guardado $May$ cambia su valor por el que se encontró. Cuando se termine de recorrer la matriz se muestra por pantalla el mayor elemento.

La subrutina Transpuesta recibe como parámetros el número de filas y de columnas de una matriz $M1$, y la matriz donde se guardará su transpuesta, $M2$. Como la transpuesta de una matriz se halla cambiando las filas por columnas, entonces a la nueva matriz en la posición $[fila, columna]$ se le asigna lo que tiene la original en $[columna, fila]$, a través de dos ciclos Para anidados que recorren la matriz.

Para buscar los elementos impares que se encuentran en la matriz, la subrutina Impar calcula el módulo de cada elemento de la matriz al dividirlo por dos (2); en caso de que sea diferente a cero, lo descarta y continúa, pero si es igual aumenta el tamaño del vector donde se guardará y lo asigna en esta posición, y sigue con el siguiente; al final del procedimiento queda guardada en $tam$ la cantidad de elementos de $V$, vector donde se almacenan los números impares que hay en la matriz. Para mostrar su contenido se hace uso de la subrutina mostrar_Vector, cuyo funcionamiento fue explicado anteriormente.

La subrutina escribir_Matriz recibe como parámetros el número de filas, de columnas y la matriz que se va a mostrar. Se recorre cada posición a través de dos ciclos Para anidados, mostrando por medio de la primitiva Escriba el contenido de la matriz en la posición $[i, j]$.

El algoritmo principal se encarga de hacer los llamados a las funciones y subrutinas, mandando los parámetros correspondientes. En primera instancia se leen las dimensiones de la matriz $A$, $n$ para el número de filas y $m$ el de las columnas; luego se llama a la subrutina encargada de leer la matriz, para luego, por medio del procedimiento mayor, buscar y mostrar por pantalla el mayor elemento que se encuentra en la matriz; en $B$ se guardará la matriz resultante de llamar a la subrutina transpuesta, para después mostrar su contenido con escribir_Matriz.

Por último se buscarán los elementos impares que se encuentran en la matriz y se almacenarán en el vector $C$ de tamaño $k$; en caso de que no se haya almacenado nada, $k$ tiene el valor de 0, dado en la subrutina, por tanto, se mostrará un mensaje indicando que no se encontraron números impares.

**Proc**: Elemento mayor, transpuesta e impares
  **Entero:** $n$, $m$, $A[50, 50]$, $B[50, 50]$, $C[50]$, $k$
  **Lea** $n, m$
  leer_Matriz$(n, m, A)$
  mayor$(n, m, A)$
  transpuesta$(n, m, A, B)$
  escribir_Matriz$(n, m, B)$
  impar$(n, m, A, C, k)$
  **Si**$(k > 0)$**Entonces**
    escribir_Vector$(k, C)$
1 2

```
1 2
  Sino
    Escriba "La matriz no tiene números impares"
  F.Si
Fin_proc
leer_Matriz(fila, columna, M)
  Entero: i, j
  Para(i = 1, fila, 1)Haga
    Para(j = 1, columna, 1)Haga
      Escriba "Digite el elemento de la posición ", i, ", ", j
      Lea M[i, j]
    Fin_Para
  Fin_Para
Fin_leer_Matriz
mayor(fila, columna, M)
  Entero: May, i, j
  May ← 0
  Para(i = 1, fila, 1)Haga
    Para(j = 1, columna, 1)Haga
      Si(M[i, j] > May)Entonces
        May ← M[i, j]
      F.Si
    Fin_Para
  Fin_Para
  Escriba "El elemento mayor de la matriz es: ", May
Fin_mayor
transpuesta(fila, columna, M1, M2)
  Entero: i, j
  Para(i = 1, fila, 1)Haga
    Para(j = 1, columna, 1)Haga
      M2[i, j] ← M1[j, i]
    Fin_Para
  Fin_Para
Fin_transpuesta
impar(fila, columna, M, V, tam)
  Entero: tam, i, j
  tam ← 0
  Para(i = 1, fila, 1)Haga
    Para(j = 1, columna, 1)Haga

1 2 3
```

```
1 2 3
      Si(M[i, j] mod 2 ≠ 0)Entonces
         tam ← tam + 1
         V[tam] ← M[i, j]
      F.Si
    Fin_Para
  Fin_Para
Fin_impar
```

```
escribir_Matriz(fila, columna, M)
  Entero: i, j
  Para(i = 1, fila, 1)Haga
    Para(j = 1, columna, 1)Haga
      Escriba "EL elemento de la posición ", i, ", ", j, " es: ", M[i][j]
    Fin_Para
  Fin_Para
Fin_escribir_Matriz
```

```
escribir_Vector(tam, v)
  Entero: i
  Para(i = 1, tam, 1)Haga
    Escriba "El elemento en la posición ", i, "es: ", v[i]
  Fin_Para
Fin_escribir_Vector
```

De ahora en adelante las funciones leer_Matriz y escribir_Vector se considerarán entendidas.

---

Ejemplo 5.15 Eliminar repetidos

Dado un vector de enteros de tamaño $n$, diseñe un algoritmo que elimine aquellos valores que se repitan, dejando solo una ocurrencia. Por ejemplo, si la entrada es $A = [3, 5, 6, 3, 2, 4, 4, 6]$, la subrutina elemIguales debe volver el vector $A$ un nuevo vector $A$ sin elementos repetidos, es decir, $A = [3, 5, 6, 2, 4]$.

*Análisis*: Para leer y escribir un vector se hace uso de las funciones explicadas anteriormente: leer_Vector, escribir_Vector.

La función elemIguales consiste en comparar cada elemento con los demás. Si encuentra alguno igual, y no está en la última posición, entonces entra al ciclo Para más interno, controlado por $r$, y realiza corrimiento, asignándole el siguiente a la posición donde se encuentra, hasta que llegue al final; una vez termine el corrimiento, regresa al ciclo Para externo, controlado por $j$, y continúa buscando otro igual a el mismo; cuando finaliza, se regresa y continúa con la siguiente posición (avanza $i$) y realiza el mismo proceso.

El algoritmo principal solo trabaja con dos variables, un vector, $a$, y su

respectivo tamaño. Primero se lee el vector, por medio de una subrutina; luego se llama a elemIguales($n$, $a$), que es el procedimiento encargado de buscar los elementos iguales y eliminarlos; por último se muestra el vector sin números repetidos.

```
Proc: Elementos Iguales
| Entero: n, a[50]
| Lea n
| leer_Vector(n, a)
| elemIguales(n, a)
| escribir_Vector(n, a)
Fin_proc

elemIguales(tam, v)
| Entero: i, j, r
| i ← 1
| Mq(i <= tam)Haga
| | Para(j = i + 1, tam, 1)Haga
| | | Si(v[i] = v[j])Entonces
| | | | Si(j <> tam)Entonces
| | | | | Para(r = j + 1, tam, 1)Haga
| | | | | | v[j] ← v[r]
| | | | | | j ← j + 1
| | | | | Fin_Para
| | | | F.Si
| | | | tam ← tam − 1
| | | F.Si
| | Fin_Para
| | i ← i + 1
| Fin_Mq
Fin_elemIguales
```

---

Ejemplo 5.16 Operaciones con matrices $B^t$ y $B^2$

Sea la matriz $A$ de tamaño $N1 \times M1$ y la matriz $B$ de tamaño $N2 \times M2$, halle la suma de los elementos de la matriz $A$ y la suma de los elementos de la matriz $B$. Si la suma de los elementos de la matriz $A$ es mayor que la suma de los elementos de la matriz $B$, halle la transpuesta de $A$; en caso contrario halle $B^2$.

*Análisis*: Las subrutinas leer_Matriz, escribir_Matriz y transpuesta son las mismas explicadas en el ejercicio 3.

Para la suma de los elementos de una matriz, la subrutina suma recibe como parámetros una matriz, $M$, su número de filas, $fila$, y su número de columnas, *columna*. La matriz se recorre con dos Para anidados que utilizan como índices las variables enteras $i, j$ y se almacena en otra, *sum*, la suma de los elementos de la matriz.

Como lo indica su nombre, la subrutina multiplicar es la encargada de realizar esta operación de una matriz con ella misma; para esto recibe dos matrices: la original, $M1$, con su número de filas, $fila$, y columnas, $columna$, y otra para almacenar el resultado, $M2$.

Para poder multiplicar dos matrices es necesario que el número de filas de la primera sea igual al número de columnas de la segunda, por tanto, en el algoritmo principal, antes de llamar a esta subrutina hay que validar si $fila$ es igual a $columna$. El elemento correspondiente a la posición $[i, j]$ de la matriz resultante proviene de la sumatoria de multiplicar cada elemento de la fila $i$ de la primera con los de la columna $j$ de la segunda, por lo que para ir avanzando se hace uso de otra variable $k$.

El algoritmo principal emplea dos matrices que serán leídas por pantalla, $A, B$, pidiendo el número de filas y columnas de cada una, $N1, M1, N2, M2$, respectivamente. Para cada una se calcula la suma de sus elementos y se almacenan en $sumA, sumB$; luego estos valores serán comparados: si la suma de $A$ sea mayor que la de $B$, se llama a los procedimientos que calculan la transpuesta de $A$ y la muestra por pantalla; en caso contrario, y siempre y cuando esté permitido, es decir, si el número de columnas es igual al número de filas, se multiplicará $B$ con sigo misma, almacenándose en la matriz $D$, para por último mostrarla.

```
Proc: Transpuesta o producto
| Entero: N1, M1, N2, M2
| Entero: A[50, 50], B[50, 50], sumA, sumB, C[50, 50], D[50, 50]
| Lea N1, M1, N2, M2
| leer_Matriz(N1, M1, A)
| sumA ← suma(N1, M1, A)
| leer_Matriz(N2, M2, B)
| sumB ← suma(N2, M2, B)
| Si(sumA > sumB)Entonces
| | transpuesta(N1, M1, A, C)
| | escribir_Matriz(M1, N1, C)
| Sino
| | Si(N2 = M2)Entonces
| | | multiplicar(N2, M2, B, D)
| | | escribir_Matriz (N2, M2,D)
| | Sino
| | | Escriba "B^2 no se puede hallar"
| | F.Si
| F.Si
Fin_proc
```

```
suma(fila, columna, M)
| Entero: sum, i, j
| sum ← 0
| Para(i = 1, fila, 1)Haga
| | Para(j = 1, columna, 1)Haga
| | | sum ← sum + M[i, j]
| | Fin_Para
| Fin_Para
| Devuelva sum
Fin_suma

multiplicar(fila, columna, M1, M2)
| Entero: i, j, acumula
| Para(i = 1, fila, 1)Haga
| | Para(j = 1, columna, 1)Haga
| | | acumula ← 0
| | | Para(k = 1, columna, 1)Haga
| | | | acumula ← acumula + M1[i, k] * M1[k, j]
| | | Fin_Para
| | | M2[i, j] ← acumula
| | Fin_Para
| Fin_Para
Fin_multiplicar
```

---

Ejemplo 5.17 Número capicúa

---

Escriba una función que determine si un número es *capicúa* (un *número capicúa* es aquel que se lee igual de derecha a izquierda y viceversa, por ejemplo: 12321).

*Análisis*: Si se tiene el número 1234, luego se asigna a $numero2$, $numero$ **mod** 10, es decir, $numero2 = 4$, luego se divide $numero$ entre 10, quedando simplemente 123, y se realiza el mismo proceso, solo que $numero2$ se multiplica por 10 y luego se le añade el residuo módulo 10 de $numero$. Esto invierte el número; en el último caso queda un número de dos cifras, entonces se divide entre 10 para obtener las centenas, y finalmente se agrega a $numero2$; por eso en el Mientras que está la condición de $numero2 \geq 10$. Luego se le hace una copia a la variable $numero$ en $aux$. Finalmente, si $aux = numero2$, entonces el número invertido es el mismo que el número normal, por tanto, sería *capicúa* y la función booleana devuelve verdadero; en caso contrario, falso.

Suponga que el número inicial es 1234, entonces esto es lo que sucede con las variables $num1$ y $num2$.

| | num1 | num2 |
|---|---|---|
| | 1234 | 4 |
| 1 | 123 | 4 |
| | 123 | 43 |
| 2 | 12 | 43 |
| | 12 | 432 |
| 3 | 1 | 432 |
| | 1 | 4321 |

**InvertirNumero(**$num1$**)**
  **Entero:** $aux$, $num2$
  $aux \leftarrow num1$
  $num2 \leftarrow num1$ **mod** 10
  **Mq**($numero \geq 10$)**Haga**
    $num1 \leftarrow num1/10$
    $num2 \leftarrow num2 * 10 + num1$ **mod** 10
  **Fin_Mq**
  **Si**($num2 = aux$)**Entonces**
    Devuelva $verdadero$
  **Sino**
    Devuelva $falso$
  **F.Si**
**Fin_InvertirNumero**

---

Ejemplo 5.18 Coordenadas polares a rectangulares

Escriba un programa que utilice una función para convertir coordenadas polares a rectangulares.

$$x = r\cos\theta$$
$$y = r\sin\theta$$

*Análisis*: En este algoritmo, las funciones Coseno y Seno deben estar previamente definidas. La función recibe como parámetros de entrada el radio, $r$, y el ángulo, $angulo$, y según las fórmulas, asigna a $x$ y $y$ los valores correspondientes. La función imprime en pantalla los valores de $x$ y $y$, y devuelve 0.

**Convertir_Rectangulares(**$r$, $angulo$**)**
  **Real:** $x$, $y$
  $x \leftarrow r * \cos(angulo)$
  $y \leftarrow r * \sin(angulo)$
1

1
**Escriba** "$x$: ", $x$, ", $y$: ", $y$
Devuelva 0
**Fin_Convertir_Rectangulares**

---

Ejemplo 5.19 Calendario

Escriba un algoritmo, utilizando funciones, que visualice un calendario de la forma

| D | L | M | M | J | V | S |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | | | |

El usuario indica únicamente el mes y el año.

*Análisis*: Sabemos que los días de la semana son 7 y que se repiten cada séptimo día, entonces la función *PrimerDiaMes* se encarga de encontrar el número que corresponde al primer día de la semana, y se basa en contar cuántos días hay entre la fecha introducida y un marcador puesto en el algoritmo. En este caso se puede ver que la condición es $aa$ mayor o igual que 2000; esto se debe a que el primer día del año 2000 es el marcador usado.

Basándose en eso se toman los días totales que hay entre el primer día del mes/año introducido y el marcador y se aplica la operación residuo módulo 7, según convenga. Por ejemplo, si entre una fecha $x$ y una fecha $y$ hay $z$ días (sin contar el día de la fecha $x$), siendo la fecha $y$ posterior a la fecha $x$, entonces partiendo de que se sabe el día $x$ y de que el calendario de ejemplo es domingo(0), lunes(1), martes(2), miércoles(3), jueves(4), viernes (5), sábado (6), el número del día que corresponde a la fecha $y$ sería $x + (z \bmod 7)$, y esto último calculado módulo 7 nuevamente. Por ejemplo, el día 6 de noviembre de 2009 fue viernes (5), y se quiere saber qué día fue el 25 de noviembre de 2009; como ambos días están en el mismo mes, restemos, y queda que $z = 25 - 6 = 19$, luego $z$ MOD 7 es 5, y $5 + 5$ es 10, pero 10 en residuo módulo 7 es 3, entonces caería miércoles. Luego el resto resulta en escribir el calendario con un formato de salida adecuado; esto se hace primero escribiendo la primera línea del calendario, luego el resto que sigue un patrón común, y finalmente la última línea. Esto se debe a que la primera y la última línea resultan ser las "diferentes", y para facilitar el formato de salida mejor se escriben independientemente.

Por último, se sobreentiende las funciones DiasMes y EsBisiesto. La primera devuelve los días de un mes, dado el año y el mes; la segunda devuelve si el año es bisiesto o no. Vea qué DiasMes debe llamar a EsBisiesto.

```
Proc: Calendario
| Entero: m, a
| Escriba "Ingrese mes"
| Lea m
| Escriba "Ingrese año"
| Lea a
| Calendario(m, a)
Fin_proc
EsBisiesto(A)
| Instrucciones
Fin_EsBisiesto
DiasMes(A, M)
| Instrucciones
Fin_DiasMes
PrimerDiaMes(mm, aa)
| Entero: Dias, i, y
| Dias ← 0
| Si(aa ≥ 2000)Entonces
| | Para(i = 2000, mm − 1, 1)Haga
| | | Si(EsBisiesto(i) = verdadero)Entonces
| | | | Dias ← Dias + 366
| | | Sino
| | | | Dias ← Dias + 365
| | | F.Si
| | Fin_Para
| | Para(i = 1, mm − 1, 1)Haga
| | | Dias ← Dias + DiasMes(i, aa)
| | Fin_Para
| | y ← (Dias − 1) mod 7
| Sino
| | Para(i = aa + 1, 1999, 1)Haga
| | | Si(EsBisiesto(i) = verdadero)Entonces
| | | | Dias ← Dias + 366
| | | Sino
| | | | Dias ← Dias + 365
| | | F.Si
| | Fin_Para
| | Para(i = mm, 12, 1)Haga
| | | Dias ← Dias + DiasMes(i, aa)
| | Fin_Para
| | y ← 6 − (Dias mod 7)
| F.Si
1
```

```
1
| Devuelva y
Fin_PrimerDiaMes

Calendario(mes, anio)
| Booleano: u
| Entero: primer_dia, j, i, d_mes, k
| u ← falso
| primer_dia ← PrimerDiaMes(mes, anio)
| j ← 0
| Escriba "D L M M J V S"
| Para(i = 0, 6, 1)Haga
| | Si(primer_dia ≠ i y u = falso)Entonces
| | | Escriba " "
| | Sino
| | | j ← j + 1
| | | u ← verdadero
| | | Escriba j, " "
| | F.Si
| Fin_Para
| Escriba " "
| d_mes ← DiasMes(mes, anio) − j
| Para(i = 1, d_mes/7, 1)Haga
| | Para(k = j + 1, j + 7, 1)Haga
| | | Escriba j, " "
| | Fin_Para
| | j ← j + 7
| | Escriba ""
| Fin_Para
| Para(i = j + 1, Mes(mes, anio), 1)Haga
| | Escriba i, " "
| Fin_Para
Fin_Calendario
```

Ejemplo 5.20 Punto en el interior de un triángulo

Escriba una función que calcule si un punto se encuentra dentro de un triángulo del que se conocen las coordenadas de sus tres vértices.

*Análisis*: Sean $(x_1, y_1)$ las coordenadas del vértice $A$, $(x_2, y_2)$ las coordenadas del vértice $B$, $(x_3, y_3)$ las coordenadas del vértice $C$ y $(p, q)$ las coordenadas del vértice que se desea saber si está en el interior del triángulo o no. La orientación de un triángulo $ABC$ (importa el orden de las letras) con las coordenadas dadas se define como el signo de la siguiente expresión:

$$O = (x_1 - x_3)(y_2 - y_3) - (y_1 - y_3)(x_2 - x_3)$$

Luego, si se tiene un punto $O$ y tres puntos, $A, B, C$, el número de triángulos que se pueden formar es 4 (escogemos tres puntos de cuatro para formar un triángulo, o bien escogemos el punto que no va hacer parte del triángulo, de esto último habrían 4 posibilidades). Luego, si el punto $P$ está dentro del triángulo, entonces la orientación de los triángulos $ABC$, $ABP$, $BCP$ y $CAP$ debe ser la misma. Defínase la función Orientación, que devuelve falso si el signo es negativo y verdadero si el signo es positivo.

**Orientacion(**$A1X, A1Y, A2X, A2Y, A3X, A3Y$**)**
| **Real:** $v$
| $v \leftarrow (A1X - A3X) * (A2Y - A3Y)$
| $v \leftarrow v - (A1Y - A3Y) * (A2X - A3X)$
| **Si**$(v \geq 0)$**Entonces**
| | $Orientacion \leftarrow verdadero$
| **Sino**
| | $Orientacion \leftarrow falso$
| **F.Si**
**Fin_Orientacion**

Luego, entonces, a partir de esta función creamos una que nos dirá si un punto está en el interior de un triángulo o no. $Ax, Ay, Bx, By, Cx, Cy, Px, Py$ son las coordenadas $x, y$ de los puntos $A, B, C$ y $P$, respectivamente.

**SIEsta_NOEsta(**$Ax, Ay, Bx, By, Cx, Cy, X, Y$**)**
| **Booleano:** $O1$, $O2$, $O3$, $O4$
| $O1 \leftarrow$ Orientacion$(Ax, Ay, Bx, By, Cx, Cy)$
| $O2 \leftarrow$ Orientacion$(Ax, Ay, Bx, By, X, Y)$
| $O3 \leftarrow$ Orientacion$(Bx, By, Cx, Cy, X, Y)$
| $O4 \leftarrow$ Orientacion$(Cx, Cy, Ax, Ay, X, Y)$
| **Si**$(O1 = O2$ y $O2 = O3$ y $O3 = O4)$**Entonces**
| | Devuelva $verdadero$
| **Sino**
| | Devuelva $falso$
| **F.Si**
**Fin_SIEsta_NOEsta**

Finalmente, llamamos en el programa principal a la función principal y con eso concluimos la solución.

---

Ejemplo 5.21 Máximo común divisor

Escriba un algoritmo que permita el cálculo del máximo común divisor (mcd) sin usar el algoritmo de Euclides y mediante el uso de funciones. Calcule igualmente el mínimo común múltiplo (mcm). Si se tienen dos números, $m$ y $n$,

$$m = {p_1}^{\alpha_1} {p_2}^{\alpha_2} \cdots {p_k}^{\alpha_k}$$
$$n = {p_1}^{\beta_1} {p_2}^{\beta_2} \cdots {p_k}^{\beta_k}$$

donde los $\beta_i$ y los $\alpha_i$ pueden ser 0, entonces el máximo común divisor de $m$ y $n$, representado como $\text{mcd}(m, n)$, sería

$$\text{mcd}(m, n) = {p_1}^{min(\alpha_1,\beta_1)} {p_2}^{min(\alpha_2,\beta_2)} \cdots {p_k}^{min(\alpha_k,\beta_k)}$$

Y el mínimo común múltiplo estaría dado por

$$\text{mcm}(m, n) = {p_1}^{max(\alpha_1,\beta_1)} {p_2}^{max(\alpha_2,\beta_2)} \cdots {p_k}^{max(\alpha_k,\beta_k)}$$

**MCD(**$a, b$**)**
  **Entero:** $mcd$, $i$
  $mcd \leftarrow 1$
  $i \leftarrow 2$
  **Mq**($a > 1$ y $b > 1$ y $i \leq a$ y $i \leq b$)**Haga**
    **Mq**($a$ **mod** $i = 0$ y $b$ **mod** $i = 0$)**Haga**
      $a \leftarrow a/i$
      $b \leftarrow b/i$
      $mcd \leftarrow mcd * i$
    **Fin_Mq**
    $i \leftarrow i + 1$
  **Fin_Mq**
  Devuelva $mcd$
**Fin_MCD**

Luego, el mínimo de dos números más el máximo de esos mismos dos números es simplemente la suma de los dos, $min(a, b) + max(a, b) = a + b$; este resultado es obvio, ya que uno de los dos es el máximo y el otro el mínimo. Partiendo de que

$$m \times n = ({p_1}^{\alpha_1} {p_2}^{\alpha_2} \cdots {p_k}^{\alpha_k})({p_1}^{\beta_1} {p_2}^{\beta_2} \cdots {p_k}^{\beta_k})$$
$$= {p_1}^{\beta_1+\alpha_1} {p_2}^{\beta_2+\alpha_2} \cdots {p_k}^{\beta_k+\alpha_k} = \prod_{i=1}^{k} {p_i}^{\beta_i+\alpha_i}$$
$$= \prod_{i=1}^{k} {p_i}^{min(\beta_i,\alpha_i)+max(\beta_i,\alpha_i)} = mcd(m, n) \times mcm(m, n)$$

Entonces $m \times n = mcm(m, n) \times mcd(m, n)$. Luego, calcule el mínimo común múltiplo resulta de dividir el producto $m \times n$ entre el máximo común divisor de $m$ y $n$. Entonces:

**MCM**$(a, b)$
**Entero:** $mcd$, $mcm$
$mcd \leftarrow MCD(a, b)$
$mcm \leftarrow a * b/mcd$
Devuelva $mcm$
**Fin_MCM**

---

Ejemplo 5.22 Primos circulares

El número 197 es llamado un *primo circular* porque todas las rotaciones de los dígitos: 197, 971, y 719, también son primos. Hay 13 números primos menores que 100: 2, 3, 5, 7, 11, 13, 17, 31, 37, 71, 73, 79, y 97, que son primos circulares. ¿Cuántos primos circulares hay menores que un millón?

*Análisis*: Hay dos posibles enfoques para este problema. El primero es ir revisando de número en número y viendo cuáles son primos, y luego verificando la condición de primos circulares. Sin embargo, observe que cuando se verifica si son primos circulares también se hace uso de si son primos. El segundo enfoque consiste en generar primero a través de la Criba de Erastótenes todos los primos menores que un millón. De tal forma que cuando se desea verificar si un número es primo solo se tiene que ver si está en el vector, la posición corresponde a 0 o a 1.

Para el primer enfoque se utilizará una función Meter. Esta convertirá un número entero en un arreglo; esto facilitará las rotaciones del número. En esta función se va extrayendo cada una de las cifras del número y se van colocando en un arreglo. Por el contrario, la función Devolver vuelve el arreglo un número entero. La función Meter recibe al vector $a$, el número que va a colocar en él y el tamaño $nc$. Dado que $nc$ se modifica en el interior de la función Meter, entonces $nc$ corresponde al número de cifras del número o tamaño del vector $a$.

La función Rotar realiza rotaciones sobre el arreglo $a$. Una vez que hace la rotación, llama a la función Devolver, que vuelve este arreglo un número. Y una vez devuelto, se llama a la función EsPrimo.

**Proc**: Primos circulares
**Entero:** $c$, $nc$, $i$, $a[50]$
$c \leftarrow 0$
$nc \leftarrow 0$
**Para**$(i = 2, 1000000, 1)$**Haga**
**Si**(EsPrimo$(i)$ = $verdadero$)**Entonces**
$nc \leftarrow 0$
Meter$(i, a, nc)$
1 2 3

```
1 2 3
| | | Si(Rotar(a, nc) = verdadero)Entonces
| | | | c ← c + 1
| | | F.Si
| | F.Si
| Fin_Para
| Escriba "Respuesta: ", c
Fin_proc

Meter(i, a, nc)
| Entero: ci
| ci ← i
| Mq(ci >= 1)Haga
| | nc ← nc + 1
| | a[nc] ← ci mod 10
| | ci ← ci/10
| Fin_Mq
Fin_Meter

Devolver(a, nc)
| Entero: num, i
| num ← 0
| Para(i = nc, 1, −1)Haga
| | num ← num * 10 + a[i]
| Fin_Para
| Devuelva num
Fin_Devolver


Rotar(a, nc)
| Booleano: sw
| Entero: i, aux, j
| sw ← verdadero
| Para(i = 1, nc − 1, 1)Haga
| | aux ← a[i]
| | Para(j = 2, nc, 1)Haga
| | | a[j − 1] ← a[j]
| | Fin_Para
| | a[nc] ← aux
| | sw ← EsPrimo(Devolver(a, nc))
| | Si(sw = falso)Entonces
| | | i ← nc + 1
| | F.Si
| Fin_Para
| Devuelva sw
Fin_Rotar
```

El segundo enfoque es completamente igual al primero. Sin embargo, considere a 'EsPrimo' no como una función, sino como un vector que contiene solamente ceros o unos. Por último, en total hay 55 números que cumplen la propiedad antes mencionada.

---

Ejemplo 5.23 Palíndromes, o capicúas, en base 2 y base 10

---

El número decimal $585 \equiv 1001001001_2$ (binario) es palíndrome en ambas bases. Encuentre la suma de todos los números enteros menores que un millón que son palíndromes en base 10 y base 2. (Note que un número palíndrome, en cualquier base, no puede incluir ceros a derecha).

*Análisis*: Primero se revisa si un número es palíndrome o no en base 10. Esto lo hará la función Palíndrome. Esta devuelve falso si el número no es palíndrome y verdadero en caso contrario. Para ello, primero cuenta las cifras de un número $n$. Suponga que se numeran las cifras de un número, de derecha a izquierda. Por ejemplo, si $n = 7453$, entonces la primera cifra es 3, la segunda 5, la tercera 4 y la cuarta 7. Observe que la cifra $i$-ésima está dada por $(n/10^{i-1})\,\mathbf{mod}\,10$. Ahora bien, mire que para que un número de cuatro cifras sea palíndrome la cifra 1 debe ser igual a la cifra 4, la cifra 2 debe ser igual a la cifra 3. Es decir, si $d$ es el número de cifras, entonces la cifra $i$ debe ser igual a la cifra $d-i+1$. Sin embargo, la cifra en la posición $d-i+1$ es $(n/10^{d-i+1-1})\,\mathbf{mod}\,10 = (n/10^{d-i})\,\mathbf{mod}\,10$. Es decir, para $i = 1, 2, 3, \cdots, \frac{d}{2}$ se tiene que cumplir que $(n/10^{i-1})\,\mathbf{mod}\,10 = (n/10^{d-i})\,\mathbf{mod}\,10$.

Además de lo anterior se debe convertir el número $n$ a base 2; para ello se utiliza la subrutina Base2. Esta subrutina no solo convierte el número a base 2, sino que también lo introduce en el arreglo $a$ y determina el tamaño de este en $nc$. Una vez se tiene esto, la función Palindrome2 verifica si este arreglo es palíndrome o no. Es decir, si el número en la posición $i$ es igual al número en la posición $nc-i+1$. Por último se cuentan los números que cumplen esta propiedad.

La función LlenarCeros inicia todas las posiciones del vector en 0 nuevamente, de tal manera que se utiliza el mismo arreglo $a$ y no se crea uno nuevo en cada iteración.

**Proc**: Palindromes en Base 2 y Base 10
  **Entero:** $c$, $nc$, $i$, $a[50]$
  $c \leftarrow 0$
  $nc \leftarrow 0$
  **Para**$(i = 1, 1000000, 1)$**Haga**
    **Si**(Palindrome($i$) $=$ *verdadero*)**Entonces**
      LlenarCeros($a, nc$)
      Base2($i, a, nc$)
1 2 3

```
1 2 3
| | | Si(Palindrome2(a, nc) = verdadero)Entonces
| | | | c ← c + 1
| | | F.Si
| | | nc ← 0
| | F.Si
| Fin_Para
| Escriba "Respuesta: ", c
Fin_proc

ContarCifras(n)
| Entero: cn, sum
| cn ← n
| sum ← 0
| Mq(cn ≥ 1)Haga
| | cn ← cn/10
| | sum ← sum + 1
| Fin_Mq
| Devuelva sum
Fin_ContarCifras

LlenarCeros(a, nc)
| Entero: i
| Para(i = 1, nc, 1)Haga
| | a[i] ← 0
| Fin_Para
Fin_LlenarCeros

Palindrome(num)
| Entero: num2, d, i
| Booleano: sw
| num2 ← num, sw ← verdadero
| d ← ContarCifras(num)
| Para(i = 1, d/2, 1)Haga
| | Si((num/10^(i−1) mod 10 ≠ (num/10^(d−i))) mod 10)Entonces
| | | sw ← falso
| | | i ← d + 1
| | F.Si
| Fin_Para
| Devuelva sw
Fin_Palindrome

Palindrome2(a, nc)
| Entero: i
| Booleano: sw
| sw ← verdadero
1
```

```
1
  Para(i = 1, nc/2, 1)Haga
    Si(a[i] ≠ a[nc − i + 1])Entonces
      sw ← falso
      i ← nc
    F.Si
  Fin_Para
  Devuelva sw
Fin_Palindrome2

Base2(num, a, nc)
  Entero: num2, i, num3, p, d
  num2 ← num
  i ← 0
  num3 ← 0
  i ← i − 1
  p ← 1
  Mq(p ≤ num2)Haga
    p ← p ∗ 2
    i ← i + 1
  Fin_Mq
  p ← p/2
  num2 ← num2 − p
  a[1] ← 1
  d ← i
  nc ← d + 1
  Mq(num2 > 0)Haga
    i ← i − 1, p ← 1
    Mq(p ≤ num2)Haga
      p ← p ∗ 2
      i ← i + 1
    Fin_Mq
    p ← p/2
    num2 ← num2 − p
    Si(i > 0)Entonces
      a[d − i + 1] ← 1
    F.Si
  Fin_Mq
  Si(num2 mod 2 = 1)Entonces
    a[nc] ← 1
  F.Si
Fin_Base2
```

En total, el número de números menores que 1 000 000 que cumplen que son palíndromes en base 10 y base 2 son 872187.

Ejemplo 5.24 Números 9-pandigital

Tome el número 192 y multiplíquelo por 1, 2 y 3:

$$\begin{aligned}192 \times 1 &= 192\\ 192 \times 2 &= 384\\ 192 \times 3 &= 576\end{aligned}$$

Por concatenación de cada producto se obtiene 192384576; este número es un 9-pandigital (puesto que se forma con una permutación de las cifras del número 123456789). Nosotros llamamos a 192384576 el producto concatenado de 192 y (1, 2, 3). Lo mismo sucede iniciando con el número 9 y multiplicando por 1, 2, 3, 4, y 5. Esto resulta en el pandigital 918273645, el cual es la concatenación del producto de 9 y $(1, 2, 3, 4, 5)$. ¿Cuál es el más grande 9-pandigital de 9-dígitos que puede ser formado como la concatenación de los productos de un entero con $(1, 2, \ldots, n)$, donde $n > 1$?

*Análisis*: Note que si un número $n$ tiene $k$ cifras, entonces $2n$ tiene al menos $k$ cifras también. Por lo tanto, la concatenación de $n$ y $2n$ tiene al menos $2k$ cifras. En este caso, $2k \leq 9$, lo que implica que $k \leq 4$. Es decir, el máximo entero que se tomará es 9999. En la función principal un ciclo externo Para irá desde 2 hasta 9999. No se tiene en cuenta el 1, puesto que es claro que el número formado por $1 \cdot 1$, $1 \cdot 2$, ..., $1 \cdot 9$ es 123456789. A su vez, se declara *max* en 123456789, puesto que el menor 9-pandigital de 9 dígitos es 123456789. Observe que cualquier otra permutación de las cifras de este número produce un número mayor que él.

Ahora bien, la función Meter es la misma del ejemplo 10. Cada vez que añade un número, este se añade al final y, a su vez, incrementa el tamaño de *nc*. Solo se añade un número en caso de que *nc* sea menor que 9. Si se llega a superar este valor, el ciclo Para interno se termina. Una vez que este se termina, se verifica si $nc = 9$, puesto que es posible que su valor se haya excedido.

Para verificar que el número sea 9-pandigital se organizan mediante el método de Burbuja los números en el arreglo. Una vez ordenadas las cifras, si el número es 9-pandigital, el número en la posición $i$ del vector debe ser $i$. De no serlo, el número no es 9-pandigital. Otra opción posible es determinar si ningún número se repite y si todos son diferentes de 9.

**Proc**: Productos concatenados
  **Entero:** *max*, *nc*, *i*, *j*, *a*[50], *d*
  **Booleano:** *sw*
  $max \leftarrow 123456789$
  $nc \leftarrow 0$
1

```
1
    Para(i = 2, 9999, 1)Haga
        nc ← 0
        Para(j = 1, 10, 1)Haga
            Si(nc < 9)Entonces
                Meter(i ∗ j, a, nc)
            Sino
                d ← j − 1
                j ← 11
            F.Si
        Fin_Para
        Si(nc = 9)Entonces
            Burbuja(a, nc)
            sw ← Verificar(a)
            Si(sw = verdadero)Entonces
                nc ← 0
                Para(j = 1, d, 1)Haga
                    Meter(i ∗ j, a, nc)
                Fin_Para
                Si(Devolver(a, nc) > max)Entonces
                    max ← Devolver(a, nc)
                F.Si
            F.Si
        F.Si
    Fin_Para
    Escriba "Respuesta: ", max
Fin_proc

Verificar(a)
    Booleano: sw
    Entero: i
    sw ← verdadero
    Para(i = 1, 9, 1)Haga
        Si(a[i] ≠ i)Entonces
            i ← 10
            sw ← falso
        F.Si
    Fin_Para
    Devuelva sw
Fin_Verificar
```

El mayor número 9-pandigital es 932718654.

Ejemplo 5.25 Primos especiales

El número 3797 tiene una interesante propiedad: es primo, y es posible remover continuamente dígitos de izquierda a derecha y ver que en cada número restante es primo: 3797, 797, 97 y 7. Igualmente se hace de derecha a izquierda, y los números obtenidos también son primos: 3797, 379, 37 y 3.

Encuentre la suma de los 11 primos que tienen esta propiedad tanto de derecha a izquierda como de izquierda a derecha. **Nota**: 2, 3, 5 y 7 no son incluidos.

*Análisis*: Dado que se tratará con números primos, se debe disponer de una función que evalúe si un número es primo con un vector generado por la Criba de Erastótenes. Se utilizará un ciclo Mientras que, el cual termina una vez se hayan encontrado los 11 primos que cumplen la propiedad. Se declara un contador inicial $i$ en 11 como el primer primo mayor que 7. De esta forma, hasta que no se encuentren los 11 números primos que cumplen la propiedad, el ciclo no termina. El procedimiento toma el valor de $i$, verifica las propiedades antes mencionadas y si encuentra que ambas se cumplen, entonces incrementa en 1 el contador $p$, si no incrementa en 2 el valor de $i$. Ahora bien, es claro que dentro de las cifras del número $i$ no debe existir el 2, excepto la primera. Por lo tanto, primero se verifica si tiene 2 o no dentro de sus cifras. De no tener 2 dentro de sus cifras, se verifica las dos propiedades antes mencionadas.

La función PDer verifica que se cumpla la propiedad de que cuando se eliminan cifras por derecha el número resultante es primo. La función PIzq hace lo mismo, pero por izquierda.

En el caso de PDer, para eliminar la última cifra basta con dividir entre 10. En el caso de PIzq es un poco más complicado. Sin embargo, en vez de ir eliminando cifras por izquierda, lo que se debe hacer es ir agregando cifras por la izquierda. Observe que al ir eliminando por izquierda, la última cifra es $n \bmod 10$. Las últimas dos cifras son $n \bmod 100$, y así sucesivamente. Por lo tanto, mientras que para PDer se usa $n/10^i$, para PIzq se usa $n \bmod 10^i$.

```
Proc: Primos extraños
| Entero: i, p, sum
| i ← 11
| p ← 0
| sum ← 0
| Mq(p < 11)Haga
| | Si((i − 9) mod 10 = 0)Entonces
| | | i ← i + 4
1 2 3
```

```
1 2 3
    Sino
      Si((i − 1) mod 10 = 0 o (i − 5) mod 10 = 0)Entonces
        i ← i + 2
      F.Si
    F.Si
    Si(Tiene2(i) = falso)Entonces
      Si(PDer(i) = verdadero y PIzq(i) = verdadero)Entonces
        Si(EsPrimo(i) = verdadero)Entonces
          sum ← sum + i
          p ← p + 1
        F.Si
      F.Si
    F.Si
    i ← i + 2
  Fin_Mq
  Escriba "Respuesta: ", sum
Fin_proc

Tiene2(num)
  Entero: num2,
  Booleano: sw, sw2
  num2 ← num
  sw ← falso
  Mq(num2 > 0 y sw = falso)Haga
    sw2 ← (num2 mod 2 = 0 y num2 > 2) o (num2 = 1)
    Si(sw2 o ((num2 − 5) mod 10 = 0 y num2 > 5))Entonces
      sw ← verdadero
    F.Si
    num2 ← num2/10
  Fin_Mq
  Devuelva sw
Fin_Tiene2

PDer(num)
  Entero: num1
  Booleano: sw
  num1 ← num
  sw ← verdadero
  Mq(num1 > 10 y sw = verdadero)Haga
    num1 ← num1/10
    sw ← EsPrimo(num1)
  Fin_Mq
  Devuelva sw
Fin_PDer
```

```
PIzq(num)
| Entero: num1, p
| Booleano: sw
| num1 ← num
| p ← 10
| sw ← verdadero
| Mq(p < num y sw = verdadero)Haga
| | num1 ← num mod p
| | p ← p * 10
| | sw ← EsPrimo(num1)
| Fin_Mq
| Devuelva sw
Fin_Entero
```

La suma de los 11 números primos que cumplen esta propiedad es 748317.

---

Ejemplo 5.26 Partes fraccionarias

La parte fraccionaria de un número irracional es creada por la concatenación de números enteros positivos:

$$0.123456789101\boldsymbol{1}12131415161718192021...$$

Es posible ver que el dígito 12 de la parte fraccionaria es 1. Si $d_n$ representa el $n$-ésimo dígito de la parte fraccionaria, encuentre el valor de la siguiente expresión:

$$d_1 \times d_{10} \times d_{100} \times d_{1000} \times d_{10000} \times d_{100000} \times d_{1000000}$$

*Análisis*: Los valores de $d_1, d_{10}, d_{100}, ..., d_{1000000}$ se pueden calcular fácilmente. Por ejemplo, $d_{10000}$; véase que entre 1 y 999 hay 2889 dígitos $(9 + 90 \times 2 + 900 \times 3)$. Entonces $10000 - 2889$ corresponde a los dígitos que van desde el número 1000 hasta el dígito $d_{10000}$ de algún número $n > 1000$; como estos números son de 4 cifras $(10000 - 2889)/4 = 1777$, $1777 + 999 = 2776$ sería el número del cual estamos hablando, pero como la división deja residuo 3, entonces en realidad correspondería al tercer dígito del siguiente número, es decir, $d_{10000} = 7$.

$$\begin{aligned} d_1 &= 1 \\ d_{10} &= 1 \\ d_{100} &= 5 \\ d_{1000} &= 3 \end{aligned} \qquad \begin{aligned} d_{10000} &= 7 \\ d_{100000} &= 2 \\ d_{1000000} &= 1 \end{aligned}$$

Para ello se hace uso de la función Meter, que va colocando en el arreglo los números 1, 2, 3, . . ., 2000000. Dado que solo se pide hasta $d_{1000000}$, entonces hay que ir revisando que el tamaño $nc < 1000000$, para evitar cálculos no

necesarios. Por otro lado, una vez que se genera el vector que contiene el número

$$123456789101112131415161718192021\ldots$$

se procede a calcular el valor de la expresión.

```
Proc: Partes fraccionarias
| Entero: p, nc, i, a[2000000]
| p ← 1
| nc ← 0
| Para(i = 1, 2000000, 1)Haga
| | Si(nc < 1000000)Entonces
| | | Meter(i, a, nc)
| | Sino
| | | i ← 20000000
| | F.Si
| Fin_Para
| Para(i = 1, i ≤ 1000000, i = i * 10)Haga
| | p ← p * a[i]
| Fin_Para
| Escriba "Respuesta: ", p
Fin_proc
```

El valor de la expresión es 210.

---

Ejemplo 5.27 Mayor número primo pandigital

Sabemos que un número es pandigital de $n$-dígito si hace uso de todos los dígitos de 1 a $n$ exactamente una vez. Por ejemplo, 2143 es un número pandigital de 4 dígitos. ¿Cuál es el más grande primo pandigital $n$-dígito que existe?

*Análisis*: Véase que ese número está entre 1 y 987654321, pues 987654321 sería el $n$-pandigital más grande posible. Ahora bien, véase que $1 + 2 + 3 + ... + 9$ es divisible por 3 y $1 + 2 + ... + 8$ también lo es, entonces los números pandigital de 8 y 9 dígitos nunca serán primos. Por consiguiente, basta revisar entre 1 y 7654321 y hacer saltos que eviten revisar aquellos números pandigital de 2, 3, 5 y 6 dígitos, pues siempre serán divisibles por 3. Pero como 1 no es primo, entonces solo basta con revisar los números de 4 cifras y los de 7.

Las funciones Meter, Burbuja y Verificar son las mismas de ejemplos anteriores. La primera coloca un número en el arreglo y modifica la variable $nc$ con el nuevo tamaño del arreglo; la segunda ordena el vector $a$ de menor a mayor; la tercera verifica que el número en la posición $k$ del vector sea igual a $k$. Con la combinación de estas tres funciones se verifica si un número es pandigital o no.

Teniendo en cuenta lo anterior se deben recorrer todos los números entre $i = 1234$ y $i = 7654321$. Para cada uno de ellos se verifican las propiedades en orden de costo computacional. Por lo tanto, primero se verifica si es pandigital o no y luego se verifica si el número es primo o no. Esto último lo hace la función EsPrimo. Recuerde que puede haber dos enfoques: el primero es que ‘EsPrimo’ es una función que recibe un número $n$ y devuelve si es primo o no, y el segundo es que ‘EsPrimo’ es un vector lleno de ceros y unos, donde si en la posición $i$ hay 1 significa que el número $i$ es primo.

Se declara inicialmente la variable $max$ en 0; si se encuentra un valor mayor, se actualiza el valor de esta variable con este nuevo valor.

```
Proc: Números primos pandigital
| Entero: max, i, nc, a[500]
| max ← 0
| Para(i = 1234, 7654321, 1)Haga
| | Si((i > 999 y i ≤ 4321) o (i > 999999 y i ≤ 7654321))Entonces
| | | nc ← 0
| | | Meter(i, a, nc)
| | | Burbuja(a, nc)
| | | Si(Verificar(a, nc) = verdadero)Entonces
| | | | Si(EsPrimo(i) = verdadero)Entonces
| | | | | Si(i > max)Entonces
| | | | | | max ← i
| | | | | F.Si
| | | | F.Si
| | | F.Si
| | Sino
| | | i ← 1234567
| | F.Si
| Fin_Para
| Escriba "Respuesta: ", max
Fin_proc
```

El mayor número primo es 7652413.

---

Ejemplo 5.28 Números triangulares, pentagonales y hexagonales

Los números triangulares, pentagonales y hexagonales son generados por la siguiente fórmula:

| | | |
|---|---|---|
| Triangular | $T_n = n(n+1)/2$ | 1, 3, 6, 10, 15, ... |
| Pentagonal | $P_n = n(3n-1)/2$ | 1, 5, 12, 22, 35, ... |
| Hexagonal | $H_n = n(2n-1)$ | 1, 6, 15, 28, 45, ... |

Es posible ver que $T_{285} = P_{165} = H_{143}$. Encuentre el siguiente número triangular que también es pentagonal y hexagonal.

*Análisis*: Se debe crear una función Hex, que reciba como entrada un número *num* y su salida sea el número hexagonal correspondiente. Dos funciones adicionales verifican si el número es triangular o no (EsTrian) y si el número es pentagonal o no (EsPent). Supongamos que se tiene un número hexagonal Hex(*num*). Se tiene que verificar que las ecuaciones

$$\text{Hex}(num) = \frac{n(n+1)}{2}, \quad \text{Hex}(num) = \frac{n(3n-1)}{2}$$

tienen soluciones enteras en $n$. Para ello se despeja $n$ de cada una y se deja en función de Hex(*num*). Si al remplazar Hex(*num*) se obtiene un $n$ entero, entonces quiere decir que el número es triangular (en el caso de la primera) y pentagonal (en el caso de la segunda).

Por último, vea que

$$\frac{n(n+1)}{2} = m \to n = \frac{-1 + 2\sqrt{2m + \frac{1}{4}}}{2}$$

y,

$$\frac{n(3n-1)}{2} = m \to n = \frac{1 + 2\sqrt{6m + \frac{1}{4}}}{6}$$

Estas fórmulas se incluyen en las funciones EsTrian y EsPent para verificar que los números sean triangulares y pentagonales, respectivamente.

**EsTrian**($m$)
| **Real:** $n$
| $n \leftarrow (-1 + 2 * \sqrt{2 * m + 0.25})/2$
| **Si**($n = \lfloor n \rfloor$)**Entonces**
| | Devuelva *verdadero*
| **Sino**
| | Devuelva *falso*
| **F.Si**
**Fin_EsTrian**

**EsPent**($m$)
| **Real:** $n$
| $n \leftarrow (1 + 2 * \sqrt{6 * m + 0.25})/6$
| **Si**($n = \lfloor n \rfloor$)**Entonces**
| | Devuelva *verdadero*
| **Sino**
| | Devuelva *falso*
| **F.Si**
**Fin_EsPent**

**Hex**($num$)
| Devuelva $num * (2 * num - 1)$
**Fin_Hex**

**Proc**: Números triangulares, pentagonales y hexagonales
| **Entero:** $i$
| **Booleano:** $sw$
| $i \leftarrow 144$
| $sw \leftarrow falso$
| **Mq**($sw = falso$)**Haga**
| | **Si**(EsPent(Hex($i$)) $= verdadero$ y EsTrian(Hex($i$)) $= verdadero$)**Entonces**
| | | $sw \leftarrow verdadero$
| | | **Escriba** "Respuesta: ", $Hex(i)$
| | **F.Si**
| | $i \leftarrow i + 1$
| **Fin_Mq**
**Fin_proc**

El mayor número que cumple las propiedades antes mencionadas es 1533776805.

---

Ejemplo 5.29 Conjetura falsa de Goldbach

---

Christian Goldbach propuso que cada número impar compuesto puede ser escrito como la suma de un primo y el doble de un cuadrado.

Tal conjetura resultó falsa.

¿Cuál es el menor número impar compuesto que no puede ser escrito como la suma de un primo y el doble de un cuadrado?

$$\begin{aligned} 9 &= 7 + 2 \times 1^2 \\ 15 &= 7 + 2 \times 2^2 \\ 21 &= 3 + 2 \times 3^2 \\ 25 &= 7 + 2 \times 3^2 \\ 27 &= 19 + 2 \times 2^2 \\ 33 &= 31 + 2 \times 1^2 \end{aligned}$$

*Análisis*: Se utiliza un contador $i$ que irá de número impar compuesto en número impar compuesto. Para verificar si un número es compuesto se hace uso de la función EsPrimo. Nuevamente existen dos enfoques: puede que EsPrimo sea una función o un arreglo que contenga solamente ceros o unos. Si EsPrimo($i$) es falso, entonces el número $i$ es compuesto. Si se inicia el contador $i$ en 7 y se incrementa de 2 en 2 en cada iteración, entonces se está buscando todos los números impares.

Una vez que se valida que un número sea impar y compuesto, se debe verificar si se puede escribir como la suma de un número primo y el doble de un cuadrado perfecto. Para esto existen dos opciones: o restarle al número $i$ el doble de un cuadrado perfecto y verificar que el resultado sea primo o restarle a un número $i$ un número primo y verificar que el resultado sea un cuadrado perfecto. Se implementará la segunda opción. Se invita al lector a implementar la primera.

Un ciclo externo Mientras que irá de impar compuesto en impar compuesto; este se cierra cuando un switch $sw$ toma el valor de verdadero. Lo que se hará es que apenas se encuentre un número impar compuesto, este $sw$ se pone en verdadero, y solo se pondrá en falso si se encuentra una descomposición, es decir, si se encuentra que el número $i$ es la suma de un número primo y el doble de un cuadrado perfecto.

Por último se utiliza la función IsSquare para verificar si un número entero $n$ es cuadrado perfecto o no.

```
Proc: Conjetura de Goldbach
| Entero: i, j
| Booleano: sw
| i ← 7
| sw ← falso
| Mq(sw = falso)Haga
| | i ← i + 2
| | Si(EsPrimo(i) = falso)Entonces
| | | // Se asume que no se encontrará un número primo
| | | // y un cuadrado perfecto
| | | sw ← verdadero
| | | Para(j = 2, i − 2, 1)Haga
| | | | Si(EsPrimo(j))Entonces
| | | | | Si((i − j) mod 2 = 0)Entonces
| | | | | | Si(IsSquare((i − j)/2) = verdadero)Entonces
| | | | | | | // Se encontró una pareja
| | | | | | | sw ← falso
| | | | | | F.Si
| | | | | F.Si
| | | | F.Si
| | | Fin_Para
| | F.Si
| Fin_Mq
| Escriba "Respuesta: ", i
Fin_proc
```

**IsSquare**($num$)
  **Si**($\sqrt{num} = \lfloor\sqrt{num}\rfloor$)**Entonces**
    Devuelva *verdadero*
  **Sino**
    Devuelva *falso*
  **F.Si**
**Fin_IsSquare**

El menor número impar compuesto que no puede ser escrito como la suma de un primo y el doble de un cuadrado es 5777.

---

Ejemplo 5.30 Números enteros consecutivos con factores primos distintos

Los dos primeros números enteros consecutivos que tienen dos primos factores distintos son: $14 = 2 \times 7$ y $15 = 3 \times 5$. Los primeros tres números enteros consecutivos que tienen 3 factores primos distintos son: $644 = 2^2 \times 7 \times 23$, $645 = 2 \times 5 \times 43$, $646 = 2 \times 17 \times 19$. Encuentre los primeros cuatros enteros consecutivos que tienen cuatro factores primos distintos. ¿Cuál es el primero de ellos?

*Análisis*: La función Factores cuenta el número de factores primos distintos de un número. En su interior tiene dos ciclos Mientras que anidados. El externo busca un factor y el interno divide al número por este factor tantas veces sea necesario. De esta forma, no es necesario verificar si al dividir el número se está dividiendo por un número primo. Cuando encuentra un divisor, incrementa el valor de $f$ en 1.

En el procedimiento principal se hace el uso de esta función para determinar si cuatro números consecutivos tienen cada uno 4 factores primos distintos. Apenas encuentra el número cambia el valor de *sw* a falso, de tal manera que se sale del ciclo Mientras que.

**Proc**: Factores primos
  **Entero:** $i$
  **Booleano:** $sw$
  $i \leftarrow 0$
  $sw \leftarrow falso$
  **Mq**($sw = falso$)**Haga**
    $i \leftarrow i + 1$
    **Si**(Factores($i$) = 4 y Factores($i+1$) = 4)**Entonces**
      **Si**(Factores($i+2$) = 4 y Factores($i+3$) = 4)**Entonces**
        $sw \leftarrow falso$
      **F.Si**
    **F.Si**
  **Fin_Mq**
1

```
1
| Escriba "Respuesta: ", i
Fin_proc

Factores(num)
| Entero: i, c, f
| i ← 2
| c ← 0
| f ← 0
| Mq(i ≤ num)Haga
| | c ← 0
| | Mq(num mod i = 0 y num > 0)Haga
| | | c ← c + 1
| | | num ← num/i
| | Fin_Mq
| | Si(c > 0)Entonces
| | | f ← f + 1
| | F.Si
| | i ← i + 1
| Fin_Mq
| Devuelva f
Fin_Factores
```

El primer número que cumple la propiedad deseada es 134043.

---

Ejemplo 5.31 Número especial

---

La secuencia aritmética 1487, 4817, 8147, en la cual cada uno de los términos es incrementado por 3330, es inusual por dos cosas: 1) cada uno de los tres términos es un número primo, y 2) cada uno de los números es una permutación de los 4 dígitos de otro. No hay secuencia aritmética con esta propiedad para números primos de 1, 2 o 3 dígitos, pero hay otra que posee esta propiedad además de la planteada. ¿Cuál es el número de 12 dígitos que se forma por la concatenación de los tres términos de esta secuencia (en orden de menor a mayor)?

*Análisis*: Debido a que se forma un número de 12 dígitos, entonces el número inicial es de cuatro cifras. Un primer ciclo externo buscará la primera cifra; un segundo ciclo externo buscará la segunda, con la restricción de que sea mayor que el primero. De tal forma que el primer ciclo va con $i$ desde 1000 hasta 9998 y el segundo va con $j$ desde $i+1$ hasta 9999. Una vez se tienen los dos primeros es posible determinar el tercero. Recuerde que si $a, b, c$ están en progresión aritmética, entonces $b - a = c - b \rightarrow c = 2b - a$; por lo tanto, el tercer número es $2b - a$.

Primero se verifica que $i$ y $j$ sean números primos, luego que $2j - i < 9999$ y finalmente que $2j - i$ sea también un número primo. Una vez que

se tienen tres números, $i, j, 2j - i$, en progresión aritmética, siendo todos primos se debe verificar que cada uno tenga las mismas 4 cifras. La función VerficarPerm se encargará de esto. Para esto, dicha función coloca cada uno de los tres números dentro de tres vectores mediante la subrutina Meter. Se ordena cada vector mediante la subrutina Burbuja y por último se verifica que los tres vectores resultantes, $id1$, $id2$ y $id3$, sean iguales.

```
Proc: Secuencia poco común
| Entero: i, j
| Para(i = 1000, 9998, 1)Haga
| | Para(j = i + 1, 9999, 1)Haga
| | | Si(i ≠ 1487)Entonces
| | | | Si(EsPrimo(i) = verdadero y EsPrimo(j) = verdadero)Entonces
| | | | | Si(2 * j - i < 9999)Entonces
| | | | | | Si(EsPrimo(2 * j - i) = verdadero)Entonces
| | | | | | | Si(VerificarPerm(i, j))Entonces
| | | | | | | | Escriba "Respuesta: ", i, j, 2 * j - i
| | | | | | | | i ← 10000, j ← i
| | | | | | | F.Si
| | | | | | F.Si
| | | | | F.Si
| | | | F.Si
| | | F.Si
| | Fin_Para
| Fin_Para
Fin_proc
```

```
VerificarPerm(x, y)
| Entero: nc1, nc2, nc3, x1, x2, x3, id1[50], id2[50], id3[50], i, c
| nc1 ← 0, nc2 ← 0, nc3 ← 0
| x1 ← x, x2 ← y
| x3 ← 2 * y - x
| Meter(x1, id1, nc1)
| Meter(x1, id3, nc2)
| Meter(x1, id2, nc3)
| Burbuja(id1, nc1)
| Burbuja(id2, nc2)
| Burbuja(id3, nc3)
| c ← 0
| Para(i = 1, 4, 1)Haga
| | // Se verifica que los tres vectores sean iguales
| | Si(id1[i] = id2[i] y id2[i] = id3[i])Entonces
| | | c ← c + 1
| | F.Si
| Fin_Para
1
```

1
**Si**$(c = 4)$**Entonces**
Devuelva $verdadero$
**Sino**
Devuelva $falso$
**F.Si**
**Fin_VerificarPerm**

El número formado es 296962999629.

---

Ejemplo 5.32 Número primo como la suma de primos consecutivos

El número primo 41 puede ser escrito como la suma de seis números primos consecutivos: $41 = 2 + 3 + 5 + 7 + 11 + 13$. Esta es la más grande suma de números primos consecutivos que se pueden añadir menor que 100. La más grande suma de números primos consecutivos menor que 1000 contiene 21 términos, y es igual a 953. Determine el número primo, menor que un millón, que puede ser escrito como la suma de más primos consecutivos.

*Análisis*: Se usará la función SigPrimo, la cual calcula el menor primo mayor un número entero, $n$, que recibe como entrada . Si este es 1, tal función asigna a $i$ el valor de 2; si este es 2, asigna a $i$ el valor de 3, y si no asigna a $i$ el valor de $n+2$. Se verifica si $i$ es primo, y si no se incrementa en 2 hasta que lo sea. Finalmente, se devuelve $i$. SigPrimo utiliza la función EsPrimo.

En el procedimiento principal se tiene una variable $pi$, que será el primo inicial, y una variable $p$, que inicia en $pi$. $p$ es una variable que se va acumulando en $sum$. Cuando $sum$ es un número primo, entonces se almacena tanto el número de primos que se sumaron como el valor de esta. Una vez que no se cumple la condición $sum + \text{SigPrimo}(p) < 1000000$, $sw$ toma el valor de verdadero y el ciclo interno Mientras que se cierra. Cuando este se cierra, si el número de primos que se sumaron es mayor que una variable $maxc$, se actualiza el valor de esta y el valor de la suma correspondiente en $mayor$. Luego se cambia a $pi$ por el menor primo mayor que él, es decir, a $pi$ se le asigna SigPrimo($pi$). Si su valor es mayor que 1000000, entonces se termina el ciclo externo Mientras que. Por último se escribe la respuesta.

**Proc**: Suma de primos consecutivos
**Entero:** $p$, $mayor$, $maxc$, $c$, $pi$, $sum$, $sumt$
**Booleano:** $sw$, $sw2$
$p \leftarrow 2, mayor \leftarrow 0, maxc \leftarrow 0, c \leftarrow 0$
$sw \leftarrow false, sw2 \leftarrow false$
$pi \leftarrow 2$
**Mq**$(sw2 = falso)$**Haga**
$c \leftarrow 1$
1 2

```
1 2
    p ← pi
    sum ← pi
    sw ← falso
    Mq(sw = falso)Haga
      Si(sum + SigPrimo(p) < 1000000)Entonces
        p ← SigPrimo(p)
        sum ← sum + p
        c ← c + 1
        Si(EsPrimo(sum) = verdadero)Entonces
          sumt ← sum
          pt ← c
        F.Si
      Sino
        sw ← verdadero
      F.Si
    Fin_Mq
    Si(pt > maxc)Entonces
      maxc ← pt
      mayor ← sumt
    F.Si
    pi ← SigPrimo(pi)
    Si(pi > 1000000)Entonces
      sw2 ← verdadero
    F.Si
  Fin_Mq
  Escriba "Respuesta: ", mayor
Fin_proc

SigPrimo(n)
  Entero: i
  Si(n = 1)Entonces
    i ← 2
  Sino
    Si(n = 2)Entonces
      i ← 3
    Sino
      i ← n + 2
    F.Si
  F.Si
  Mq(EsPrimo(i) = falso)Haga
    i ← i + 2
  Fin_Mq
  Devuelva i
Fin_SigPrimo
```

La suma buscada es 997651.

---

Ejemplo 5.33 Combinaciones

Hay exactamente 10 formas de seleccionar 3 dígitos en 12345: 123, 124, 125, 134, 135, 145, 234, 235, 245 y 345. En combinatoria, nosotros usamos la notación $5C_3 = 10$. En general

$$nC_r = \frac{n!}{r!(n-r)!}, \quad \text{donde } r \leq n, n! = n \times (n-1) \times ... \times 3 \times 2 \times 1, \text{ y } 0! = 1$$

Determine para cuántos valores, no necesariamente distintos, de $nC_r$, con $1 \leq n \leq 100$, se tiene que $nC_r$ es más grande que un millón.

*Análisis*: Se crea una función Comb, que determinará el valor de ${}^nC_r$. Esta, a su vez, hace el uso de la función Fact para calcular el factorial de un número. En el procedimiento principal se utilizan dos ciclos Para: el externo, que va desde $i = 1$ hasta $i = 100$ (este es el valor de $n$), y el interno, que va desde $j = 1$ hasta $i$ (este es el valor de $r$). El ciclo interno va hasta $i$ porque $r \leq n$. En la variable $t$ se va contando cada vez que Comb$(i, j) > 1000000$.

```
Comb(c, r)
| Devuelva Comb
| Factorial(c)/(Factorial(r) * Factorial(c − r))
Fin_Comb

Proc: Combinatorias
| Entero: t, i, j
| t ← 0
| Para(i = 1, 100, 1)Haga
| | Para(j = 1, i, 1)Haga
| | | Si(Comb(i, j) > 1000000)Entonces
| | | | t ← t + 1
| | | F.Si
| | Fin_Para
| Fin_Para
| Escriba "Respuesta: ", t
Fin_proc
```

El número de números que cumplen es 5070.

Ejemplo 5.34 Factorización prima

Escriba un algoritmo que lea un entero positivo y a continuación llame a una subrutina o función que visualice la factorización prima del número.

*Análisis*: El algoritmo principal lee *num* y luego llama a la función Factorizacion para que esta muestre la factorización prima del número.

Si se quiere determinar la factorización prima de un número, se puede iniciar en 2, y luego, si el número es divisible por 2, entonces dividirlo tantas veces como sea necesario, luego contar cuántas veces se dividió e ir anotando al estilo $2^{veces}$, luego repetir el proceso para 3, después para 4; ahora bien, 4 no es primo, pero sus divisores o factores primos ya los hemos contado antes, pues vamos en orden, entonces no será divisible por 4, y no podrá realizar el proceso (en esto se basa el método de encontrar primos mediante la Criba de Erastótenes), pero esto se puede evitar preguntando si el número es primo mediante la función EsPrimo, que ya hemos usado con anterioridad. Sin embargo, esta función EsPrimo realizará mucho más proceso que simplemente preguntar si el número es divisible o no. Entonces, es posible usar una función llamada SigPrimo y devolver el valor que sigue después de un número primo, o ir preguntando número por número. Ambas son soluciones válidas. Para mayor profundización, a continuación se presenta una solución en la que se usa la función SigPrimo; ahora bien, como el número 2 se trató aparte, entonces el contador $i$ inicia en 3, entonces el mínimo valor que devolverá la función SigPrimo, será 5, pues es el primo que sigue a 3. Esta función usa dentro de ella la función EsPrimo, pues irá preguntando de 2 en 2, de impar en impar, por si es primo; en caso de encontrar 1, devuelve inmediatamente ese valor. Se pregunta siempre si $cont > 0$ para evitar que escriba durante la factorización prima potencias nulas, es decir, $3^0$, $5^0$, $7^0$.

Por otro lado, cualquier factor primo de $num2$ será menor o igual que su raíz.

**Proc**: Factorización prima
| **Entero:** *num*
| **Escriba** "Ingrese numero"
| **Lea** *num*
| Factorizacion(*num*)
**Fin_proc**

**Factorizacion(***num***)**
| **Entero:** $num2$, $i$, $cont$
| $num2 \leftarrow num, i \leftarrow 3, cont \leftarrow 0$
| **Mq**($num2$ **mod** $2 = 0$)**Haga**
| | $num2 \leftarrow num2/2$
1 2

```
1 2
|   | cont ← cont + 1
|   Fin_Mq
|   Si(cont > 0)Entonces
|   | Escriba "2ˆ", cont
|   F.Si
|   Mq(i ≤ √num2)Haga
|   | Si(EsPrimo(i) = verdadero)Entonces
|   | | cont ← 0
|   | | Mq(num2 mod i = 0)Haga
|   | | | num2 ← num2/i
|   | | | cont ← cont + 1
|   | | Fin_Mq
|   | | Si(cont > 0)Entonces
|   | | | Escriba " * ", i, "∧", cont
|   | | F.Si
|   | F.Si
|   | i ← i + 2
|   Fin_Mq
Fin_Factorizacion
```

Como se mencionó anteriormente, si se divide el número por 3 tantas veces como lo permita, entonces este número no puede ser divisible por 9, ni por 12, ni por cualquier múltiplo de 3. Por lo tanto, cada vez que se divida por el número siempre se hará por un factor primo. Siguiendo esta lógica, entonces es posible eliminar la condición de si $i$ es primo. Este enfoque se muestra a continuación. De hecho, este segundo enfoque resulta menos costoso computacionalmente. Vea que en vez de revisar si $i$ es primo, solamente se va a revisar si $num2$ es divisible por $i$.

```
Factorizacion(num)
|   Entero: num2, i, cont
|   num2 ← num, i ← 3, cont ← 0
|   Mq(num2 mod 2 = 0)Haga
|   | num2 ← num2/2
|   | cont ← cont + 1
|   Fin_Mq
|   Si(cont > 0)Entonces
|   | Escriba "2ˆ", cont
|   F.Si
|   Mq(i ≤ √num2)Haga
|   | cont ← 0
|   | Mq(num2 mod i = 0)Haga
|   | | num2 ← num2/i
|   | | cont ← cont + 1
|   | Fin_Mq
1 2
```

```
1 2
    Si(cont > 0)Entonces
        Escriba " * ", i, "∧", cont
    F.Si
    i ← i + 2
  Fin_Mq
Fin_Factorizacion
```

Por último, también es posible usar la función SigPrimo, de tal forma que siempre se está buscando divisores primos. Se puede notar que $i$ se asigna como SigPrimo($i$), entonces $i$ incrementará de primo en primo. Sin embargo, este último enfoque realiza un mayor número de iteraciones y cálculos que la solución anterior.

```
Factorizacion(Entero: num)
  Entero: num2, i, cont
  num2 ← num, i ← 3, cont ← 0
  Mq(num2 mod 2 = 0)Haga
    num2 ← num2/2
    cont ← cont + 1
  Fin_Mq
  Escriba "2∧", cont
  Mq(i ≤ num2)Haga
    cont ← 0
    Mq(num2 mod i = 0)Haga
      num2 ← num2/i
      cont ← cont + 1
    Fin_Mq
    Si(cont > 0)Entonces
      Escriba " * ", i, "∧", cont
    F.Si
    i ← SigPrimo(i)
  Fin_Mq
Fin_Factorizacion
```

---

Ejemplo 5.35 Números amigables

Se define $d(n)$ como la suma de los divisores propios de $n$ ($i$ es un divisor propio si $i$ divide a $n$ y $i < n$). Si $d(a) = b$ y $d(b) = a$, donde $a \neq b$, entonces $a$ y $b$ son una pareja amigable y son llamados *números amigables*. Por ejemplo, los divisores propios de 220 son 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 y 110, entonces $d(220) = 284$. Los divisores propios de 284 son 1, 2, 4, 71 y 142, así que $d(284) = 220$. Encuentre la suma de los números amigables menores que 10000.

*Análisis*: $d(a) = b$ (1) y $d(b) = a$ (2), entonces remplazando 1 en 2 nos queda $d(b) = d(d(a)) = a$, pero los números perfectos (aquellos tales que $d(n) = n$)

cumplirán que $d(d(a)) = a$, pues no importa cuántas veces se aplique la operación $d(n)$ siempre dará el mismo resultado. Ejemplo: 28, sus divisores propios suman 28, entonces $d(28) = 28$, luego $d(d(28)) = d(28) = 28$, luego $d(\cdots d(d(28))\ldots) = 28$. Por lo tanto, se tendría que $d(d(a)) = a$ y las soluciones serían $(a, b) = (a, d(a))$, pero recordemos que $a \neq b$, y como los números perfectos cumplen que $d(d(n)) = n$ (es decir, se tendría $a = b$), entonces la condición sería todos los números tales que $d(d(n)) = n$ y $d(n) \neq n$.

Se crea una función llamada SDP (suma de divisores propios); esta revisa si un número es divisor de *num*, y si lo es, acumula en la variable *sum*. Como se trata de los divisores propios, lo hace solamente para divisores menores que *num*, es decir, menores o iguales que $num/2$. Al final devuelve la suma de los divisores.

Finalmente se pregunta con un contador $i$ desde 1 hasta 9999, y si $i$ cumple, entonces se suma a $c$ el valor de $i$, siendo $c$ un acumulador de suma que fue declarado en 0 debido a que es el módulo de la suma.

```
SDP(num)
|  Entero: sum, i
|  sum ← 0
|  Para(i = 1, num/2, 1)Haga
|  |  Si(num mod i = 0)Entonces
|  |  |  sum ← sum + i
|  |  F.Si
|  Fin_Para
|  Devuelva sum
Fin_SDP

Proc: Suma de divisores
|  Entero: c, i
|  c ← 0
|  Para(i = 1, 9999, 1)Haga
|  |  Si(i = SDP(SDP(i)) y SDP(i) ≠ i)Entonces
|  |  |  c ← c + i
|  |  F.Si
|  Fin_Para
|  Escriba c
Fin_proc
```

La suma buscada es 31626.

---

Ejemplo 5.36 Fórmula de Euler para generar primos

Euler publicó la notable fórmula cuadrática

$$n^2 + n + 41$$

Resulta que esta fórmula produce 40 primos consecutivos para los valores desde $n = 0$ a 39. Sin embargo, cuando $n = 40$, $40^2+40+41 = 40(40+1)+41$ es divisible por 41m y, ciertamente, cuando $n = 41$, $41^2 + 41 + 41$ es claramente divisible por 41. Con utilización de computadores se descubrió que la increíble fórmula $n^2 - 79n + 1601$ produce 80 primos para los valores consecutivos de $n = 0$ a 79. El producto de los coeficientes, $-79$ y 1601, es -126479. Teniendo en cuenta las ecuaciones cuadráticas de la forma

$$n^2 + an + b, \quad \text{donde } |a| < 1000 \text{ y } |b| < 1000$$

encuentre el producto de los coeficientes, $a$ y $b$, para la expresión cuadrática que produce el máximo de números primos para valores de $n$ consecutivos, comenzando en $n = 0$.

*Análisis*: Como el enunciado menciona que se encontraron hasta 80 primos aproximadamente, entonces fijado un $i$ y un $j$ se puede poner en un ciclo Para que vaya desde 0 hasta 80 y pregunte si la expresión $n^2 + in + j$ es primo, luego, eso conlleva al anidamiento de 3 ciclos Para. Un ciclo Para que cuente todo los valores de $a$, otro todos los de $b$ y otro los de $n$; ahora bien, en caso de no encontrar un primo, se designa un valor a $k$ lo suficiente grande como para que se salga del ciclo Para. Si es $c$ es mayor que $m$ (la variable que almacena el mayor), entonces se asigna a $m$ un nuevo valor, $c$.

```
Proc: Contraejemplo de la fórmula de Eueler
  Entero: m, i, j, c, k, p
  m ← 0
  Para(i = −999, 999, 1)Haga
    Para(j = −999, 999, 1)Haga
      c ← 0
      Para(k = 0, 80, 1)Haga
        Si(EsPrimo(k ∗ k + i ∗ k + j)Entonces
          c ← c + 1
        Sino
          k ← 200
        F.Si
      Fin_Para
      Si(c > m)Entonces
        m ← c
        p ← i ∗ j
      F.Si
    Fin_Para
  Fin_Para
  Escriba p
Fin_proc
```

El contraejemplo buscado es $a = -61$ y $b = 971$, $ab = -59231$.

Ejemplo 5.37 Número triangular con más de 500 divisores

La secuencia de números triangulares es generada sumando los números naturales. El séptimo número triangular sería $1+2+3+4+5+6+7=28$. Los primeros 10 números triangulares son:

$$1, 3, 6, 10, 15, 21, 28, 36, 45, 55, \cdots$$

En la lista de los divisores de los primeros números triangulares que se muestra a continuación se puede ver que el 28 es el primer triangular que tiene más de 5 divisores. ¿Cuál es el valor del primer número triangular que tiene más de 500 divisores?

**1**: 1
**3**: 1, 3
**6**: 1, 2, 3, 6
**10**: 1, 2, 5, 10
**15**: 1, 3, 5, 15
**21**: 1, 3, 7, 21
**28**: 1, 2, 4, 7, 14, 28

*Análisis*: Se utiliza una función (Div o Div2) para contar el número de divisores de un número. La función Div se diferencia de la función Div2 debido a su eficiencia. La primera se basa en el hecho de que si se tiene un número $n$ factorizado, como ${p_1}^{\alpha_1}{p_2}^{\alpha_2}...p_k^{\alpha_k} = \prod_{i=1}^{k} {p_i}^{\alpha_i}$, el número de divisores estaría dado por $\tau n = (\alpha_1)(\alpha_2)...(\alpha_k)$, entonces la función se encarga de sacar cada factor primo y contar su potencia y luego ir multiplicando el número de divisores $p$ por el siguiente $e+1$.

Dado que un número triangular se forma por $T_n = n(n+1)/2$, entonces se verifica si el número de divisores de $T_n$ es mayor que 500. Una vez se tiene el número, el ciclo Mientras que se cierra y se imprime el resultado.

```
Proc: Número de divisores de un número triangular
| Entero: i
| Booleano: sw
| i ← 5
| Mq(sw = falso)Haga
| | i ← i + 1
| | Si(Div(i * (i + 1)/2) > 500)Entonces
| | | sw ← verdadero
| | F.Si
| Fin_Mq
| Escriba i
Fin_proc
```

```
Div(num)
| Entero: i, c, p, e
| i ← 2, e ← 1, p ← 1
| Mq(i ≤ num)Haga
| | e ← 0
| | Mq(num MOD i = 0)Haga
| | | num ← num/i
| | | e ← e + 1
| | Fin_Mq
| |
| | Si(e > 0)Entonces
| | | p ← p * (e + 1)
| | F.Si
| | i ← i + 1
| Fin_Mq
| Devuelva p
Fin_Div

Div2(num)
| Entero: p, i
| p ← 2
| Para(i = 2, num/2, 1)Haga
| | Si(num mod i = 0)Entonces
| | | p ← p + 1
| | F.Si
| Fin_Para
| Devuelva p
Fin_Div2
```

El número triangular con más de 500 divisores es 76576500.

---

Ejemplo 5.38 Suma de los términos impares de Fibonacci

Cada nuevo término en la secuencia de Fibonacci es generado por la suma de los dos anteriores términos. Empezando en 1 y 2, los 10 primeros términos serían: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89. Encuentre la suma de los términos impares en la secuencia que no exceden los cuatro millones.

*Análisis*: Se utiliza una función que genera números de Fibonacci, esta es la función Fib. Se verifica si el número generado es impar. En caso de serlo, se acumula en la variable *sum*. Finalmente se imprime el resultado.

```
Proc: Serie de Fibonacci
| Entero: f, sum, limit, i
| f ← 1, sum ← 0, limit ← 4000000, i ← 0
1
```

1
**Mq**($f < limit$)**Haga**
$i \leftarrow i + 1$
$f \leftarrow Fib(i)$
**Si**($f$ **mod** $2 = 1$)**Entonces**
$sum \leftarrow sum + f$
**F.Si**
**Fin_Mq**
**Escriba** "Respuesta: ", *sum*
**Fin_proc**

**Fib**($n$)
**Entero:** $a$, $b$, $c$, $i$
$a \leftarrow 1, b \leftarrow 1, c \leftarrow 0$
**Para**($i = 1, n, 1$)**Haga**
$c \leftarrow a + b$
$a \leftarrow b$
$b \leftarrow c$
**Fin_Para**
Devuelva $a$
**Fin_Fib**

Observe que una forma más eficiente de resolver este problema es ir generando números de Fibonacci y al tiempo ir revisando si es impar y menor que 4000000. Esta solución se presenta a continuación:

**Proc**: Segunda Solución
**Entero:** $a$, $b$, $c$, $sum$, $limit$
$a \leftarrow 1, b \leftarrow 1, c \leftarrow 0$
$sum \leftarrow 0, limit \leftarrow 4000000$
**Mq**($a < limit$)**Haga**
$c \leftarrow a + b$
$a \leftarrow b$
$b \leftarrow c$
**Si**($a$ **mod** $2 = 1$)**Entonces**
$sum \leftarrow sum + a$
**F.Si**
**Fin_Mq**
**Escriba** "Respuesta: ", *sum*
**Fin_proc**

Por último, la suma buscada es 4613732.

---

Ejemplo 5.39 Mayor factor primo de 600851775143

---

Los factores primos de 13195 son 5, 7, 13 y 29. ¿Cuál es el mayor factor primo del número 600851775143?

*Análisis*: Se debe utilizar la función EsPrimo y también se debe revisar hasta la raíz cuadrada. Vea que el número es impar, por lo tanto, el primer número primo que se revisará es 3. Por consiguiente, se incrementa este número de 2 en 2, en vez de ir de 1 en 1. Se revisa si este número es mayor que un número $max$, si divide a $n$ y si es primo. En caso de serlo, se remplaza el valor de $max$.

Por último, $EsPrimo$ puede ser o una función o un vector de ceros y unos.
**Proc**: Mayor factor primo
| **Entero:** $i$, $max$, $n$
| $i \leftarrow 1, max \leftarrow 0$
| $n \leftarrow 600851775143$
| **Mq**$(i < \sqrt{n})$**Haga**
| | $i \leftarrow i + 2$
| | **Si**$(n\, \mathbf{mod}\, i = 0$ y $i > max)$**Entonces**
| | | **Si**(EsPrimo$(i)$ = $verdadero$)**Entonces**
| | | | $max \leftarrow i$
| | | **F.Si**
| | **F.Si**
| **Fin_Mq**
**Fin_proc**

El mayor factor primo es 6857.

---

Ejemplo 5.40 Mayor palíndrome producto de dos números de 3 dígitos

---

Un número palíndrome se lee igual en ambos sentidos. El palíndrome más grande que resulta del producto de dos números de 2 dígitos es $9009 = 91 \times 99$. Encuentre el número palíndrome más grande producto de dos números de 3 dígitos.

*Análisis*: Dado que se trata del producto de dos números de 3 dígitos, entonces se utilizan dos ciclos Para (uno interno y otro externo) para verificar si el número $ij$ es palíndrome. En caso de serlo y ser mayor que $max$, se remplaza la variable $max$ (que contiene el mayor) por $ij$. Por último se escribe el resultado. Se utiliza la función EsPalindrome; esta recibe $num$, copia el valor de este en $num2$ y crea un número $num3$, que es el número $num2$ invertido. Este proceso es el mismo que el discutido en ejemplos anteriores. Finalmente, esta función verifica si $num3$ es igual a $num$.

**Proc**: Números palíndromes especiales
| **Entero:** $max$, $i$, $j$
| $max \leftarrow 0$
| **Para**$(i = 100, 999, 1)$**Haga**
| |
1 2

1 2
**Para**$(j = 100, 999, 1)$**Haga**
**Si**$(i > max$ y EsPalindrome$(i * j) = verdadero)$**Entonces**
$max \leftarrow i * j$
**F.Si**
**Fin_Para**
**Fin_Para**
**Escriba** "Respuesta: ", $max$
**Fin_proc**

**EsPalindrome(**$num$**)**
**Entero:** $num2$, $num3$
$num2 \leftarrow num, num3 \leftarrow 0$
**Mq**$(num2 > 0)$**Haga**
$num3 \leftarrow num3 * 10 + num2$ **mod** $10$
$num2 \leftarrow num2/10$
**Fin_Mq**
**Si**$(num3 = num)$**Entonces**
Devuelva $verdadero$
**Sino**
Devuelva $falso$
**F.Si**
**Fin_EsPalindrome**

El número buscado es 906609.

Ejemplo 5.41 Cuadrado de la suma − Suma de los cuadrados

La suma de los cuadrados de los primeros 10 números naturales es $1^2 + 2^2 + ... + 10^2 = 385$. El cuadrado de la suma de los 10 primeros números naturales es $(1 + 2 + ... + 10)^2 = 55^2 = 3025$. La diferencia entre la suma de los cuadrados de los primeros 10 números naturales y el cuadrado de la suma es $3025 - 385 = 2640$. Encuentre la diferencia entre la suma de los cuadrados de los primeros 100 números naturales y el cuadrado de la suma, es decir, encuentre

$$\left(\sum_{i=1}^{100} i\right)^2 - \left(\sum_{i=1}^{100} i^2\right)$$

Cree funciones para ello.

*Análisis*: Se crean dos funciones, SquareSum y SumSquare. La primera calcula el cuadrado de la suma de los números entre 1 y el número que recibe como entrada $num$; la segunda calcula la suma de los cuadrados de los números entre 1 y el número que recibe como entrada $num$.

**Proc**
**Entero:** *resp*
$resp \leftarrow$ SquareSum(100) − SumSquare(100)
**Escriba** "Respuesta: ", *resp*
**Fin_proc**

**SquareSum(***num***)**
**Entero:** *sum*, *i*
$sum \leftarrow 0$
**Para**$(i = 1, num, 1)$**Haga**
$sum \leftarrow sum + i$
**Fin_Para**
Devuelva $sum * sum$
**Fin_SquareSum**

**SumSquare(***num***)**
**Entero:** *sum*, *i*
$sum \leftarrow 0$
**Para**$(i = 1, num, 1)$**Haga**
$sum \leftarrow sum + i * i$
**Fin_Para**
Devuelva *sum*
**Fin_SumSquare**

---

Ejemplo 5.42 Primo número 10001

Listando los primeros 6 números primos se tiene: 2,3,5,7,11 y 13; podemos ver que el sexto primo es 13. ¿Cuál es el primo número 10001?

*Análisis*: Se utiliza la función EsPrimo para verificar si un número es primo o no. Cuando se encuentre uno se incrementa el valor de *cont* en 1. Cuando se llegue al 10001 se imprime el valor de *i*. Siendo este el primo número 10001. Otra forma de resolver el problema es haciendo uso de la función SigPrimo y realizando 100000 asignaciones de la forma $p$ = SigPrimo($p$).

**Proc**: El primo número 10001
**Entero:** *cont*, *i*
$cont \leftarrow 0, i \leftarrow 1$
**Mq**$(cont < 10001)$**Haga**
$i \leftarrow i + 1$
**Si**(EsPrimo($i$) = $verdadero$)**Entonces**
$cont \leftarrow cont + 1$
**F.Si**
**Fin_Mq**
1

1
| **Escriba** "Respuesta: ", $i$
**Fin_proc**

El primo número 104743.

---

Ejemplo 5.43 Semilla que genera más términos

---

La siguiente secuencia iterativa está definida para el conjunto de enteros positivos

$n \leftarrow n/2$ (si $n$ es impar)
$n \leftarrow 3n + 1$ (si $n$ es par)

Usando la regla anterior e iniciando en 13 generamos la siguiente secuencia: $13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$. Se puede ver que la secuencia (iniciando en 13, y terminando en 1) contiene 10 términos. Esto no ha sido probado aún (Problema de Collatz), pero iniciando en cualquier número, la secuencia siempre logra terminar en 1. Empezando con cualquier número, menor que un millón, ¿cuál produce la cadena más larga?

*Análisis*: Se crea una función llamada Terminos; esta recibe como parámetro de entrada un número y realiza el algoritmo mostrado en el enunciado. A su vez, va acumulando el número de términos que se obtienen hasta cuando se llega a 1. En el procedimiento principal se determina el máximo número de términos generados. La variable $max$ almacena el máximo número de términos generados y $r$ el punto de inicio de la secuencia.

**Proc**: La cadena más larga
| **Entero:** $max$, $i$, $r$
| $max \leftarrow 0$
| **Para**($i = 1, 1000000, 1$)**Haga**
| | **Si**(Terminos($i$) $> max$)**Entonces**
| | | $max \leftarrow$ Terminos($i$)
| | | $r \leftarrow i$
| | **F.Si**
| **Fin_Para**
| **Escriba** "Respuesta: ", $r$
**Fin_proc**

**Terminos(**$num$**)**
| **Entero:** $i$
| $i \leftarrow 1$
| **Mq**($num \neq 1$)**Haga**
| | **Si**($num$ **mod** $2 = 0$)**Entonces**
| | | $num \leftarrow num/2$
1 2 3

1 2 3
**Sino**
$num \leftarrow 3 * num + 1$
**F.Si**
$i \leftarrow i + 1$
**Fin_Mq**
Devuelva $i$
**Fin_Terminos**

El número inicial que genera mayor número de términos es 837799.

---

Ejemplo 5.44 Domingos inicio de mes

Teniendo en cuenta la siguiente información:

- El 1$^o$ de junio de 1990 fue lunes.
- Septiembre, abril, junio y noviembre tienen 30 días. Todos los demás meses tienen 31, excepto febrero, que tiene 28 en un año no bisiesto y 29 en uno bisiesto.

¿Cuántos domingos fueron inicio de mes durante el siglo XX (1$^o$ de enero de 1901 a 31 de diciembre de 2000)?

*Análisis*: Para la solución se consideran las funciones DiasMes y EsBisiesto. En el procedimiento principal se tienen dos ciclos Para anidados: el primero recorre todos los meses en la variable $i$ y el segundo todos los años en la variable $j$. La función PrimerDiaMes devuelve el primer día del mes. Devuelve 0 si es domingo, devuelve 1 si es lunes, devuelve 2 si es martes, y así sucesivamente. La función es la misma que la usada en el ejemplo del calendario (ejemplo 5.8).

**Proc**: Número de domingos
**Entero:** $i$, $j$, $cont$
**Para**($i = 1, 12, 1$)**Haga**
**Para**($j = 1901, 2000, 1$)**Haga**
**Si**($PrimerDiaMes(i, j) = 0$)**Entonces**
$cont \leftarrow cont + 1$
**F.Si**
**Fin_Para**
**Fin_Para**
**Escriba** "Respuesta: ", $cont$
**Fin_proc**

```
PrimerDiaMes(mm, aa)
| Entero: Dias, i, y
| Dias ← 0
| Si(aa ≥ 2000)Entonces
| | Para(i = 2000, mm − 1, 1)Haga
| | | Si(EsBisiesto(i) = verdadero)Entonces
| | | | Dias ← Dias + 366
| | | Sino
| | | | Dias ← Dias + 365
| | | F.Si
| | Fin_Para
| | Para(i = 1, mm − 1, 1)Haga
| | | Dias ← Dias + DiasMes(i, aa)
| | Fin_Para
| | y ← (Dias − 1) mod 7
| Sino
| | Para(i = aa + 1, 1999, 1)Haga
| | | Si(EsBisiesto(i) = verdadero)Entonces
| | | | Dias ← Dias + 366
| | | Sino
| | | | Dias ← Dias + 365
| | | F.Si
| | Fin_Para
| | Para(i = mm, 12, 1)Haga
| | | Dias ← Dias + DiasMes(i, aa)
| | Fin_Para
| | y ← 6 − (Dias MOD 7)
| F.Si
| Devuelva y
Fin_PrimerDiaMes
```

El número buscado es 171.

---

Ejemplo 5.45 Número perfectos que no son suma de dos números abundantes

Un *número perfecto* es aquel que la suma de sus divisores propios es exactamente igual al número. Por ejemplo, la suma de los divisores propios de 28 sería $1+2+4+7+14 = 28$, lo cual significa que 28 es un número perfecto. Un número $n$ es llamado *deficiente* si la suma de sus divisores propios es menor que $n$, y es llamado *abundante* si la suma excede $n$. Como 12 es el menor número abundante, $1 + 2 + 3 + 4 + 6 = 16$, el menor número abundante que puede ser escrito como la suma de dos números abundantes es 24. Un análisis matemático muestra que todos los números enteros mayores que 28123 pueden ser escritos como la suma de dos números abundantes. Encuentre la suma de todos los números enteros positivos que no pueden ser escritos como la suma de dos números abundantes.

*Análisis*: Se crea una función llamada Abund, que se encarga de verificar si un número es abundante o no. En su interior, esta función llama a SumDiv, la cual calcula la suma de los divisores, y si esta es mayor que el número devuelve verdadero, en caso contrario devuelve falso. Como el menor número abundante que puede ser escrito como la suma de dos números abundante es 24, se revisará la propiedad deseada a partir de 25. Un ciclo externo va de 1 en 1 hasta 28123, puesto que de 28123 en adelante los números pueden ser escritos como la suma de dos abundantes.

El ciclo interno determina todas las posibles parejas que suman $i$. Si un sumando es $j$, el otro es $i - j$. El contador $j$ inicia siempre en 12, puesto que este es el menor número abundante. Vea que si $j > i/2$, $i - j < i/2$, por lo tanto ya $j$ habrá tomado el valor de $i - j$. Por consiguiente, el contador $j$ va hasta $i/2$.

**Proc**: Números abundantes
  **Entero:** $sum$, $i$, $j$
  $sum \leftarrow 0$
  **Para**($i = 25, 28123, 1$)**Haga**
    **Para**($j = 12, i/2, 1$)**Haga**
      **Si**(Abund($i - j$) = *verdadero* y Abund($j$) = *verdadero*)**Entonces**
        $j \leftarrow i$
      **F.Si**
    **Fin_Para**
    **Si**($j = i/2$)**Entonces**
      $sum \leftarrow sum + i$
    **F.Si**
  **Fin_Para**
  $sum \leftarrow sum + 1 + 2 + 3 + ... + 23$
  **Escriba** "Respuesta: ", $sum$
**Fin_proc**

**Abund(**$num$**)**
  **Si**(SumDiv($num$) > $num$)**Entonces**
    Devuelva *verdadero*
  **Sino**
    Devuelva *falso*
  **F.Si**
**Fin_Abund**

La suma buscada es 4179871.

---

Ejemplo 5.46 Fibonacci con 1000 cifras

---

La secuencia de Fibonacci está definida por la relación recurrente $F_n = F_{n-1} + F_{n-2}$, donde $F_1 = 1$ y $F_2 = 2$. Los primeros 12 términos serían $F_1 = 1, F_2 = 1, F_3 = 2, F_4 = 3, F_5 = 5, F_6 = 8, F_7 = 13$. El séptimo término,

$F_7$, es el primer término que contiene 2 dígitos. ¿Cuál es el primer término en la secuencia de Fibonacci que contiene 1000 dígitos?

*Análisis*: Para esto veamos que si $g(n) = 1 + \log_{10} n$, entonces $g(10) = 2$ y $g(100) = 3$, y todo número entre 10 y 100 tendría a la parte entera de $f(n)$ igual a 2. Entonces hemos encontrado una función que calcula directamente el número de dígitos de un número. La función NumDig determina las cifras de un número mediante esta fórmula. En el procedimiento principal se llama a la función Fib, que calcula el Fibonacci número $i$. Una vez se determina el $i$-ésimo número de Fibonacci, su valor se pasa a la función NumDig. Cuando NumDig excede 1000 se escribe el resultado.

**NumDig(*num*)**
| Devuelva $\lfloor 1 + \log_{10} n \rfloor$
**Fin_NumDig**

**Proc**: Números de Fibonacci
| **Entero:** $i$
| $i \leftarrow 0$
| **Mq**(NumDig(Fib($i$)) $< 1000$)**Haga**
| | $i \leftarrow i + 1$
| **Fin_Mq**
| **Escriba** "Respuesta: ", $i$
**Fin_proc**

---

Ejemplo 5.47 Conjunto de términos $a^b$

Considere todas las combinaciones de enteros $a^b$ para $2 \leq a \leq 5$ y $2 \leq b \leq 5$:

$$\begin{array}{cccc} 2^2 = 4 & 2^3 = 8 & 2^4 = 16 & 2^5 = 32 \\ 3^2 = 9 & 3^3 = 27 & 3^4 = 81 & 3^5 = 243 \\ 4^2 = 16 & 4^3 = 64 & 4^4 = 256 & 4^5 = 1024 \\ 5^2 = 25 & 5^3 = 125 & 5^4 = 625 & 5^5 = 3125 \end{array}$$

Si todos estos términos son colocados en orden numérico, eliminando los repetidos, se obtiene la secuencia de 15 términos distintos: 4, 8, 9, 16, 25, 27, 32, 64, 81, 125, 243, 256, 625, 1024, 3125. ¿Cuántos términos distintos son generados por $a^b$ para $2 \leq a \leq 100$ y $2 \leq b \leq 100$?

*Análisis*: Vea que por las condiciones dadas para $a$ y para $b$ en total existen $99 \times 99$ combinaciones. De estas se eliminarán las repetidas. El procedimiento principal llena un vector *id* con todas las combinaciones. Luego se ordena este vector de tal forma que sea más fácil el conteo de los elementos repetidos. Lo anterior se realiza mediante el método Burbuja. Posteriormente se llama a la función Repetidos, que cuenta el número de elementos repetidos en el vector.

Para contar los elementos repetidos, puesto que están ordenados, el vector adquiere la siguiente forma $[a_1, a_1, \ldots, a_1, a_2, \ldots, a_2, \ldots, a_n, \ldots, a_n]$. Se toma un contador $i$, el cual se posiciona en 1 inicialmente, en una variable $c$ se cuenta cuántos elementos iguales tiene a derecha, y a su vez, se acumula esto en una variable $rep$. Luego de obtener cuántos elementos se tiene a derecha, la variable $i$ toma el valor de $i + c$. De tal manera que se posiciona en el último elemento repetido. Cuando el algoritmo vuelve a entrar al ciclo, la variable $i$ se incrementa en 1 nuevamente, y se repite el proceso. Por ejemplo, suponga que el vector resultante luego de ordenarse es $[1, 1, 1, 1, 2, 2, 2, 3, 3, 3]$. Primero se toma $i = 1$, se cuentan las veces que aparece $i$ a derecha, es decir, 3 veces. Por lo tanto, a $i$ se le asigna el valor de $i + c = 1 + 3 = 4$. A este punto $i$ tiene el valor de la posición del último 1 repetido. Cuando se vuelve a entrar al ciclo $i$ se incrementa en 1, tomando el valor de $i + 1 = 4 + 1 = 5$.

**Proc**: Combinaciones $a^b$
  **Entero:** $nc$, $i$, $j$, $id[10000]$
  $nc \leftarrow 0$
  **Para**$(i = 2, 100, 1)$**Haga**
    **Para**$(j = 2, 100, 1)$**Haga**
      $nc \leftarrow nc + 1$
      $id[nc] \leftarrow i^j$
    **Fin_Para**
  **Fin_Para**
  Burbuja$(id, nc)$
  **Escriba** "Respuesta: ", $99 * 99 -$ Repetidos$(id, nc)$
**Fin_proc**

**Repetidos(**$arreglo, nc$**)**
  **Entero:** $rep$, $i$, $c$, $j$, $arreglo[10000]$
  $rep \leftarrow 0$, $i \leftarrow i + 1$
  **Mq**$(i \leq nc - 1)$**Haga**
    $i \leftarrow i + 1$
    $c \leftarrow 0$
    **Para**$(j = i + 1, nc, 1)$**Haga**
      **Si**$(arreglo[i] = arreglo[j])$**Entonces**
        $rep \leftarrow rep + 1$
        $c \leftarrow c + 1$
      **F.Si**
    **Fin_Para**
    $i \leftarrow i + c$
  **Fin_Para**
  Devuelva $rep$
**Fin_Repetidos**

El número de elementos no repetidos de la forma $a^b$ con las condiciones del problema es 9183.

Ejemplo 5.48 Números como suma de potencias quintas de sus dígitos

Solo hay tres números que pueden ser escritos como la suma de las potencias cuartas de sus dígitos:

$$\begin{aligned} 1634 &= 1^4 + 6^4 + 3^4 + 4^4 \\ 8208 &= 8^4 + 2^4 + 0^4 + 8^4 \\ 9474 &= 9^4 + 4^4 + 7^4 + 4^4 \end{aligned}$$

Como $1^n = 1$ es una suma no incluida para ningún valor de $n$, la suma de estos números es $1634 + 8208 + 9474 = 19316$. Encuentre la suma de todos los números que pueden ser escritos como la suma de las potencias quintas de sus dígitos.

*Análisis*: Note que el problema no menciona ninguna cota superior para los números. Es claro que el algoritmo no puede revisar infinitos números, por lo tanto, es necesario encontrar una cota superior para este problema. Observe que si el número es de 7 cifras, la máxima suma de las potencias quintas de las cifras posible es $9^5 + 9^5 + 9^5 + 9^5 + 9^5 + 9^5 + 9^5 = 7 \times 9^5 = 413343$. Sin embargo, este número es de 6 cifras. Lo anterior implica que no es necesario revisar números de 7 o más cifras. Un ciclo Para externo tomará entonces valores desde 2 hasta 999999.

Para resolver el problema se usará una función llamada Dig, que extraerá el dígito en la posición $p$ (de derecha a izquierda) del número $num$. Para ello se divide el número inicialmente por $10^p$ (esta división es entera) y luego se calcula el residuo que deja al dividirse entre 10, es decir, se determina el residuo módulo 10. Con esta función es posible extraer cada una de las cifras de un número, elevarlas a la potencia quinta y sumarlas.

En el procedimiento principal $suma2$ es la variable que contiene la suma de las potencias quintas de las cifras. Observe que en el cálculo de esta se llama a la función Dig. Luego se verifica si $suma2$ es igual al contador externo $i$, que se encarga de buscar aquellos números que pueden ser escritos como la suma de las potencias quintas de sus cifras. Si se cumple la condición, en la variable $sum$ se acumula la suma de estos números.

**Proc**: Potencias cuartas de las cifras de un número

  **Entero:** $sum$, $i$, $suma2$
  $sum \leftarrow 0$
  **Para**$(i = 2, 354294, 1)$**Haga**
    $suma2 \leftarrow 0$
    **Para**$(j = 1, 6, 1)$**Haga**
      $suma2 \leftarrow suma2 + (\text{Dig}(i, j))^5$
    **Fin_Para**

1 2

```
1 2
      Si(suma2 = i)Entonces
        sum ← sum + i
      F.Si
    Fin_Para
    Escriba "Respuesta: ", sum
Fin_proc

Dig(num, p)
    Devuelva (num/10^p) mod 10
Fin_Dig
```

---

Ejemplo 5.49 Productos 9-pandigital

Se dice que un $n$-dígito es un *número pandigital* si usa todos los dígitos de 1 a $n$ exactamente una vez; por ejemplo, el 5-dígito, 15234, es un 5-pandigital. El producto 7254 es inusual, como la identidad $39 \times 186 = 7254$, contiene multiplicando, multiplicador y producto como un 9-pandigital (juntemos 39,186,7254, es decir, 391867254, y se puede ver que cada dígito del 1 al 9 ha sido usado exactamente una vez). Encuentre la suma de todos los productos en los que multiplicando/multiplicador/producto pueden ser escritos como un 9-pandigital.

*Análisis*: Antes de continuar se puede asumir que el multiplicador es mayor o igual que el multiplicando, de tal forma que no se repitan combinaciones. Sin embargo, dado que un número 9-pandigital requiere que no se repitan cifras, entonces el multiplicador debe ser necesariamente mayor que el multiplicando. Por lo tanto, el máximo valor que puede tomar el multiplicando es un número de 3 cifras, de tal forma que el multiplicador también, y por consiguiente, el producto será a lo mínimo de 3 cifras también. Ahora bien, el máximo posible para el multiplicando es 987; cualquier número mayor repetirá cifras. El mínimo valor posible para el multiplicando es 2, dado que si es 1, entonces el producto tendrá exactamente las mismas cifras que el multiplicando.

En el caso del multiplicador, este es máximo cuando el multiplicando es mínimo, es decir, cuando es 2. Vea que el multiplicador debe ser a lo mucho un número de 4 cifras. Si es de 5, el producto tendrá mínimo 5 cifras, lo cual lleva a que la concatenación tenga mínimo 10 cifras, lo cual conllevaría a que no fuese un número 9-pandigital. Ahora bien, si el multiplicando tiene 1 cifra y el multiplicador 4, entonces el producto debe tener 4 cifras. Por lo tanto, el multiplicando tiene que ser menor que 5000, pues el doble de un número mayor o igual que 5000 tiene 5 cifras. Siendo así, el multiplicando tiene que ser menor que 4987.

El producto tiene que ser de cuatro cifras. Si no lo fuese, entonces fuese a lo mucho de 3. Pero esto implica que el número de cifras totales entre el

multiplicando y el multiplicador fuese de 4, lo que resulta en un total de 7 cifras. Por lo tanto, el producto debe ser mayor o igual que el mínimo número de 4 cifras que no repite ninguna cifra, es decir, 1023.

Por último, en el procedimiento principal se utilizan dos ciclos Para: un externo para controlar el valor del multiplicando y uno interno para controlar el valor del multiplicador. Se verifica que el resultado $i * j$ sea mayor o igual que 1023. Luego se deben introducir $i$, $j$ y $i * j$ en el mismo vector para proceder a verificar si el número es 9-pandigital. Lo anterior se hace con las funciones antes usadas: LlenarCeros, Meter, Burbuja y Verificar. Si se cumple la condición, el producto se añade a un acumulador $sum$. Finalmente se imprime la respuesta.

```
Proc: Multiplicando/multiplicador/producto
  Entero: sum, i, j, nc, a[100]
  Booleano: sw
  sum ← 0
  Para(i = 2, 987, 1)Haga
    Para(j = i + 1, 4987, 1)Haga
      Si(i * j ≥ 1023)Entonces
        nc ← 0
        LlenarCeros(a, 100)
        Meter(i, a, nc)
        Meter(j, a, nc)
        Meter(i * j, a, nc)
        Burbuja(a, nc)
        sw ← Verificar(a)
        Si(sw = verdadero)Entonces
          sum ← sum + i * j
        F.Si
      F.Si
    Fin_Para
  Fin_Para
  Escriba "Respuesta: ", sum
Fin_proc
```

La suma buscada es 45228.

---

Ejemplo 5.50 Números curiosos

El 145 es un número curioso, pues $1!+4!+5! = 1+24+120 = 145$. Encuentre la suma de todos los números que son iguales a la suma de los factoriales de sus dígitos. **Nota**: como $1! = 1$ y $2! = 2$ no son sumas, no son incluidos.

*Análisis*: Observe que el problema no menciona una cota superior para la búsqueda de estos números, por lo tanto, es necesario encontrarla antes de proceder. La máxima suma posible de los factoriales de los dígitos de un

número de $k$ cifras es $k \times 9!$; este número como mínimo es $10^{k-1}$. Por lo tanto, se desea saber para qué valor de $k$, $k \times 9! < 10^{k-1}$. Se observa que para $k = 8$, $k \times 9! = 2903040$, lo cual es menor que $10^{8-1} = 10^7$.

Teniendo en cuenta lo anterior se utiliza un ciclo Para externo que va desde 3 hasta el máximo número posible de 7 cifras. Se determina la suma de los factoriales de sus dígitos con la función SumF. Esta función va extrayendo cada una de las cifras del número y va acumulando en la variable *sum* la suma de los factoriales de cada cifra. Para el cálculo de los factoriales llama a la función Factorial. Por último se devuelve el valor de *sum*.

En el procedimiento principal, si $i = \text{SumF}(i)$, entonces se entra en el condicional y se agrega a la variable *sum* el valor de $i$. Por último se imprime la respuesta.

```
Proc: Números curiosos
    Entero: i, sum
    i ← 3, sum ← 0
    Para(i = 1, 9999999, 1)Haga
        Si(i = SumF(i))Entonces
            sum ← sum + i
        F.Si
    Fin_Mq
    Escriba "Respuesta: ", sum
Fin_proc
```

```
SumF(num)
    Entero: num2, sum, a
    num2 ← num, sum ← 0
    Mq(num2 > 0)Haga
        a ← num mod 10
        num2 ← num2/10
        sum ← sum + Factorial(a)
    Fin_Mq
    Devuelva sum
Fin_SumF
```

La suma buscada es 40730.

---

## 5.6 Algoritmos propuestos

Resuelva los siguientes ejercicios trabajando con funciones y subrutinas.

1. Sean dos números cualesquiera que pertenecen a los naturales, imprima todos los números primos que se encuentran entre ellos dos.

**Entrada:** 7, 25.
**Salida:** 7, 11, 13, 17, 19, 23.

2. Desarrolle un algoritmo que halle la suma de los $N$ primeros elementos de la siguiente serie:
   $S = X - 3X^2/2 + 5X^4/6 - 7X^8/24 + 9X^{16}/120...$
   **Entrada:** Un valor entero para $N$ y uno para $X$.
   **Salida:** Cálculo obtenido al realizar la suma con el valor de $X$ proporcionado.

3. Utilice una función para verificar si un número cualquiera pertenece o no a la serie de Fibonacci.
   **Entrada:** Un entero positivo (Ej. 8).
   **Salida:** Proposición verdadera o falsa sobre su pertenencia a la serie de Fibonacci. (El número 8 sí pertenece a la serie Fibonacci).

4. Elabore un algoritmo que dado un vector muestre, en forma ordenada, cuáles de los datos son números primos.
   **Entrada:** $A = [1, 5, 8, 7, 6, 3, 11, 8, 4, 2]$.
   **Salida:** 2, 3, 5, 7, 11.

5. Dados $n$ números, imprima cuáles de ellos contienen por lo menos un 5.
   **Entrada:** 2, 5, 450, 687, 555, 4005, 67, 894, 9854.
   **Salida:** 5, 450, 555, 4005, 9854.

6. Lea un vector de enteros de tamaño $n$, y diga cuántos y cuáles de los datos son números perfectos.
   **Entrada:** $A = [6, 28, 50, 874, 496, 54, 954, 123, 8128, 258]$.
   **Salida:** Hay 4 números perfectos en el vector: 6, 28, 496, 8128.

7. Construya un algoritmo que muestre la representación de los $n$ primeros naturales en Binario.
   **Entrada:** n=5
   **Salida:** 0 = 000
   1 = 001
   2 = 010
   3 = 011
   4 = 100
   5 = 101

8. De un vector de tamaño $n$ se desea imprimir los números que no están repetidos, usando una función lógica que diga si un elemento se encuentra repetido o no.
   **Entrada:** $A = [5, 8, 0, 84, 96, 5, 96, 23, 8, 558]$.
   **Salida:** Los números que no están repetidos son: 0, 84, 23, 558.

9. Dados dos vectores, diga cuál de ellos tiene mayor cantidad de números con la propiedad de ser mayores a los demás elementos.

**Entrada:** $A = [1, 2, 8, 5, 6, 3, 6, 8, 8]$
$B = [4, 7, 7, 10, 1]$
**Salida:** $A$ tiene más elementos mayores que $B$.

10. Desarrolle un algoritmo que permita determinar de $n$ cilindros cuál tiene mayor área lateral y cuál mayor volumen, conociendo el radio ($r$) y la altura ($h$) de cada uno. Tenga en cuenta que el área ($A$) y el volumen ($V$) de un cilindro están dados, respectivamente, por $A = 2\pi rh$ y $V = \pi r^2 h$.

    **Entrada:** 1. Radio: 0.5 cm, Altura: 10 cm
    2. Radio: 0.4 cm, Altura: 12 cm
    3. Radio: 0.25 cm, Altura: 6 cm
    **Salida:** El cilindro con mayor área es el número 1.
    El cilindro con mayor volumen es el número 1.

11. Realice un algoritmo que permita ordenar ascendentemente tres números mediante un procedimiento de intercambio de dos variables.
    **Entrada:** 8, 3, 5
    **Salida:** 3, 5, 8

12. Diseñe un algoritmo que llame a la función signo($X$) y muestre:
    *a*) El signo de un número
    *b*) El signo de la función coseno.

    **Entrada:** $-60$
    **Salida:** a. El signo del número ($-60$) es $-$ (Negativo)
    b. El signo del coseno del número ($\cos(-60) = 0.5$) es + (Positivo)

13. Diseñe un algoritmo que calcule los salarios de los trabajadores de una empresa, donde cada trabajador tiene un número dado de horas trabajadas y un sueldo por hora.
    **Entrada:** 1. Horas trabajadas: 15; Sueldo por hora: 10000
    2. Horas trabajadas: 17; Sueldo por hora: 9500
    3. Horas trabajadas: 25; Sueldo por hora: 15000
    **Salida:** El salario del trabajador 1 es: $150000
    El salario del trabajador 2 es: $161500
    El salario del trabajador 3 es: $375000

14. *a*) Implemente una función con cuatro argumentos para recibir dos números racionales (cada número racional debe tener numerador y denominador) y que devuelva el resultado de la suma de dichos números como un número real.

    *b*) Modifique la función anterior para que devuelva el resultado como otro racional (numerador, denominador).

**Entrada:** 1. 4/5
2. 7/13
**Salida:** a. 1.33847
b. 87/65

15. Diseñe una función que lea una frase en mayúsculas, carácter por carácter, hasta que encuentre el carácter $*$ y devuelva al programa principal el número de ocurrencias en la frase de un determinado carácter que se le pasa a la función.
**Entrada:** Ingeniería, e
**Salida:** La e aparece 2 veces

16. Dados dos vectores, $A$ y $B$, de tamaño $n$ y $m$, respectivamente, determine cuál de los dos tiene el elemento mayor.
**Entrada:** $A = [5, 4, 5, 2, 8]$
$B = [15, 2, 23, 7]$
**Salida:** $B$ tiene el elemento mayor de los dos vectores, 23.

17. Escriba un procedimiento que dado un número en base 10 lo convierta a base 2.
**Entrada:** $A = 4$
**Salida:** 100

18. Dados dos vectores de tamaño $n$ y $m$, los cuales almacenan los códigos de los estudiantes que no aprobaron Algebra y Física, respectivamente, realice un algoritmo modular que determine aquellos estudiantes que perdieron ambas asignaturas y aquellas que solo perdieron una.
**Entrada:** $A = [7456, 2543, 5234, 9898, 8567]$
$B = [5234, 5743, 8654, 9898]$
**Salida:** Ambas perdidas: 5234, 9898
Una perdida: 7456, 2543, 5234, 5743, 8654, 9898, 8567

19. Dado un vector de $N$ elementos, calcule el número de elementos positivos, negativos y ceros que contiene.

**Entrada:** $A = [-2, 2, 5, -25, 0, 6, 0, -9]$
**Salida:** Positivos: 3
Negativos: 3
Ceros: 2

20. Escriba una función que acepte

*a*) Parámetro: $x$, siendo $x \neq 0$, y devuelva

$$\frac{1}{x^5\left(\frac{e^{1.435}}{x} - 1\right)}$$

*b*) Parámetro: $x$, $n$ y devuelva

$$x + \frac{x^n}{n} - \frac{x^{n+2}}{n+2} \quad \text{si } x \geq 0,\, n \neq 0$$

$$\frac{x^{n+1}}{n+1} - \frac{x^{n-1}}{n-1} \quad \text{si } x < 0,\, n \neq 1$$

*c*) Parámetro: $a$, $b$, $c$ y devuelva

$$Area = \sqrt{p(p-a)(p-b)(p-c)} \quad \text{donde } p = \frac{a+b+c}{2}$$

21. Escriba un algoritmo usando funciones que realicen las siguientes tareas:

    *a*) Devolver el valor del día de la semana en respuesta a la entrada de la letra inicial (mayúscula o minúscula) de dicho día. L = lunes, M = martes, W = miércoles, J = jueves, V = viernes, S = sábado, D = domingo.

    *b*) Determinar el número de días de un mes o año dado.

    **Entrada**: Día: W
    **Salida**: 03
    **Entrada**: Mes: Enero
    **Salida**: 31
    **Entrada**: Año: 2011
    **Salida**: 365

22. Escriba un algoritmo que acepte un número de tres dígitos y a continuación los muestre en palabras (modo alfabético).
    **Entrada:** 225
    **Salida:** Doscientos veinticinco.

23. Si los números del 1 al 5 son escritos en palabras: one, two, three, four, five, entonces allí hay $3 + 3 + 5 + 4 + 4 = 19$ letras usadas en total. Diga cuántas letras se utilizan si todos los números del 1 al 1000 (one thousand) inclusive son escritos en palabras.
    **Nota**: No contar espacios ni renglones. Por ejemplo, 342 (three hundred and forty-two) contiene 23 letras y 115 (one hundred and fifteen) 20.

24. El 1406357289 es un número 0 a 9-pandigital porque hace uso de cada uno de los dígitos de 0 a 9 en algún orden, es decir, una permutación de los dígitos 0123456789; pero tiene un rara e interesante propiedad de subcadenas. Sea $d_1$ el primer dígito, $d_2$ el segundo digito, y así sucesivamente del número pandigital, entonces formamos las siguientes subcadenas:

    - $d_2d_3d_4 = 406$ es divisible por 2
    - $d_3d_4d_5 = 063$ es divisible por 3

- $d_4d_5d_6 = 635$ es divisible por 5
- $d_5d_6d_7 = 357$ es divisible por 7
- $d_6d_7d_8 = 572$ es divisible por 11
- $d_7d_8d_9 = 728$ es divisible por 13
- $d_8d_9d_{10} = 289$ es divisible por 17

Encuentre la suma de todos los números 0 a 9-pandigital con esta propiedad.
**Solución**: 16695334890.

25. Los números pentagonales son generados por la fórmula,
$P_n = n(3n - 1)/2$.
Los primeros 10 números pentagonales son: 1, 5, 12, 22, 35, 51, 70, 92, 117, 145, ... Es posible ver que $P_4 + P_7 = 22 + 70 = 92 = P_8$. A su vez, la diferencia, $70 - 22 = 48$ no es un número pentagonal.
Encuentre la pareja de números pentagonales $P_j$ y $P_k$, donde $j, k \leq n$, para los cuales la suma y diferencia son números pentagonales y $D = |P_k - P_j|$ es minimizado.

26. Camila le dice a Juan un número $N$. Juan piensa cuatro números: $a, b, c, d$, tales que la suma de los dígitos de cada uno de estos números es la misma y $a + b + c + d = N$. Implemente un algoritmo que determine el mayor número que pudo haber pensado Juan dado el número que Camila le dice.
**Entrada:** $n$.
**Salida:** Máximo valor entre $a, b, c$ y $d$.

**Entrada ejemplo:** 2012.
**Salida ejemplo:** 1745.

27. Escriba un programa que solicite al usuario un carácter y que sitúe ese carácter en el centro de la pantalla. El usuario debe poder a continuación desplazar el caracter pulsando las teclas A (arriba), B (abajo), I (izquierda), D (derecha) y F (fin) para terminar.

# Bibliografía

Lipschutz, S. (1987). *Estructura de Datos. Teoría y 457 problemas resueltos. Serie Schaum en Computación.* Madrid: McGraw-Hill.

López, G., J. I. & Vega, A. (2009). *Análisis y Diseño de Algoritmos. Implementación en C y Pascal.* México, D.F.: Alfaomega.

Villalobos, J. (1996). *Diseño y manejo de estructura de datos en C.* Bogotá, D.C.: McGraw-Hill Interamericana S. A.

# Capítulo 6

# Problemas desafío

## 6.1 El problema $3n + 1$

### Enunciado

Fuente: UVA Online Judge, Problema 100

Considere el siguiente algoritmo:

1. Lea $n$
2. Escriba $n$
3. Si $n = 1$, entonces FIN
4. Si $n$ es impar, entonces $n \leftarrow 3n + 1$,
5. sino $n \leftarrow n/2$
6. Ir a 2

Dada la entrada 22, se imprimirá la siguiente secuencia de números: 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1.

Se conjetura que el algoritmo anterior siempre terminará (cuando un 1 se ha impreso) para cualquier valor de entrada integral. A pesar de la simplicidad del algoritmo, se desconoce si esta conjetura es verdadera. Se ha comprobado, sin embargo, para todos los enteros $n$ tales que $0 < n < 1000000$ (y, de hecho, para muchos números más que esto.)

Dado un $n$ de entrada, es posible determinar el número de números impresos (incluyendo al 1). Para un determinado $n$, a esto se le se llama la longitud del ciclo de $n$. En el ejemplo anterior, la longitud de ciclo de 22 es 16.

Para cualquiera de los dos números de $i$ y $j$ se debe determinar la longitud del ciclo máximo en todos los números entre $i$ y $j$.

**Entrada**

La entrada constará de una serie de pares de números enteros $i$ y $j$, un par de enteros por línea. Todos los números enteros serán menores que 1.000.000 y mayor que 0.

Usted debe procesar todos los pares de números enteros y para cada par determinar la longitud máxima del ciclo sobre todos los enteros entre $i$ y $j$, inclusive.

**Salida**

Para cada par de enteros de entrada $i$ y $j$ se debe mostrar $i$, $j$ y la longitud de ciclo máximo de enteros entre $i$ y $j$, inclusive. Estos tres números deben estar separados por al menos un espacio; a su vez, deben estar en una línea y debe haber una línea de salida para cada línea de entrada. Los enteros $i$ y $j$ deben aparecer en la salida en el mismo orden en que aparecen en la entrada y deben ser seguidos por la longitud de ciclo máximo (en la misma línea).

**Entrada Ejemplo**

1 10
100 200
201 210
900 1000

**Salida Ejemplo**

1 10 20
100 200 125
201 210 89
900 1000 174

## Análisis

Se utilizará una función que contará el tamaño del ciclo de $n$. Se llamará a esta función para todos los números entre $i$ y $j$ y se imprimirá la mayor longitud encontrada. La función que calculará la longitud del ciclo se llamará "longciclo". Internamente, la función realiza el algoritmo mostrado en el enunciado, es decir,

asigna $3n+1$ si $n$ es impar y $n/2$ en caso contrario. Para revisar si $n$ es par o no se revisa su residuo módulo 2.

## Solución

```
Proc: Problema 3n + 1
| Entero: sw, a, b, ans, i
| sw ← 0
| Mq(sw = 0)Haga
| | Lea a, b
| | Si(a ≠ 0)Entonces
| | | Para(i = min(a, b), max(a, b), 1)Haga
| | | | ans ← max(a, longciclo(i))
| | | Fin_Para
| | | Escriba a, " ", b, " ", ans
| | Sino
| | | sw ← 1
| | F.Si
| Fin_Mq
Fin_proc

max(a, b)
| Si(a > b)Entonces
| | Devuelva a
| Sino
| | Devuelva b
| F.Si
Fin_max    min(a, b)
| Si(a < b)Entonces
| | Devuelva a
| Sino
| | Devuelva b
| F.Si
Fin_min    longciclo(n)
| Entero: ans
| ans ← 0
| Mq(n > 1)Haga
| | ans ← ans + 1
| | Si(n mod 2 = 0)Entonces
| | | n ← n/2
| | Sino
| | | n ← 3n + 1
| | F.Si
| Fin_Mq
1
```

1
| Devuelva $ans + 1$
**Fin_longciclo**

# 6.2 Suma máxima

## Enunciado

Dado un array de 2 dimensiones de números enteros positivos y negativos, busque el subrectángulo con la suma más grande. La suma de un rectángulo es la suma de todos los elementos de ese rectángulo. En este problema, el subrectángulo con la suma más grande se conoce como el *subrectángulo máximo*. Un subrectángulo es cualquier contigua submatriz de tamaño $1 \times 1$ o mayor situado dentro del array. Como un ejemplo, el máximo subrectángulo de la matriz

$$\begin{matrix} 0 & -2 & -7 & 0 \\ 9 & 2 & -6 & 2 \\ -4 & 1 & -4 & 1 \\ -1 & 8 & 0 & -2 \end{matrix}$$

está en la esquina inferior izquierda:

$$\begin{matrix} 9 & 2 \\ -4 & 1 \\ -1 & 8 \end{matrix}$$

y tiene suma 15.

### Entrada y Salida

La entrada consiste de un arreglo de tamaño $N \times N$ de enteros. La entrada empieza con un entero positivo $N$ en una línea indicando el tamaño de la matriz cuadrada. Luego de esto le siguen $N^2$ enteros separados por espacios en blanco y líneas. Estos $N^2$ constituyen la matriz en el orden de filas (es decir, todos los números de la primera fila, de izquierda a derecha, y luego todos los números de la segunda fila, de izquierda a derecha, etc.). $N$ puede ser tan grande como 100. Los números en el arreglo estarán en el rango $[-127, 127]$.

### Entrada Ejemplo

4
0 -2 -7 0
9 2 -6 2

-4 1 -4 1 -1
8 0 -2

**Salida Ejemplo**

15

## Análisis

Note que un primer intento es tener dos punteros correspondientes a dos esquinas opuestas del rectángulo. Con estos dos punteros queda definido el rectángulo y posteriormente se podría sumar todos los elementos del rectángulo formado e ir revisando si esta suma es mayor que una ya encontrada. Sin embargo, esta solución no es del todo óptima.

Suponga que se tiene el siguiente rectángulo:

| | | | | | | |
|---|---|---|---|---|---|---|
| 19 | 1 | 20 | -8 | -4 | 14 | -3 |
| 16 | 2 | 7 | 11 | 0 | -19 | 16 |
| -1 | 7 | 1 | -7 | -4 | 6 | -20 |
| -9 | -1 | -2 | -15 | 7 | 4 | -3 |
| -15 | -32 | -14 | 6 | -10 | -15 | 5 |
| -6 | 11 | -15 | 5 | 16 | 5 | 1 |
| 2 | 6 | -3 | -12 | 7 | -10 | -25 |

Si se desea revisar la suma de todos los elementos del arreglo y, a su vez, la suma de todos los elementos del rectángulo señalado a continuación

| | | | | | | |
|---|---|---|---|---|---|---|
| 19 | 1 | 20 | -8 | -4 | 14 | -3 |
| 16 | 2 | 7 | 11 | 0 | -19 | 16 |
| -1 | 7 | 1 | -7 | -4 | 6 | -20 |
| -9 | -1 | -2 | -15 | 7 | 4 | -3 |
| -15 | -32 | -14 | 6 | -10 | -15 | 5 |
| -6 | 11 | -15 | 5 | 16 | 5 | 1 |
| 2 | 6 | -3 | -12 | 7 | -10 | -25 |

entonces se están sumando todos los elementos de este último rectángulo 2 veces. Una forma más eficiente sería restarle a la suma de todos los elementos del rectángulo más grande la suma de los elementos de la última columna y la última fila; esto requiere un menor número de cálculos.

Siguiendo la estrategia anterior, al mismo tiempo que se leerá el rectángulo se calculará una suma acumulada de los elementos del mismo, es decir, en la posición

$[i, j]$ del arreglo se encontrará la suma de todos los elementos del rectángulo con esquinas $[1, 1]$ y $[i, j]$. Por ejemplo, si el arreglo ingresado es como el de la entrada ejemplo, entonces una vez leído el arreglo se convertirá en

$$\begin{array}{cccc} 0 & -2 & -7 & 0 \\ 9 & 2 & -6 & 2 \\ -4 & 1 & -4 & 1 \\ -1 & 8 & 0 & -2 \end{array} \rightarrow \begin{array}{cccc} 0 & -2 & -9 & -9 \\ 9 & 9 & -4 & -2 \\ 5 & 6 & -11 & -8 \\ 4 & 13 & -4 & -3 \end{array}$$

Con este nuevo rectángulo formado resulta más sencillo revisar la suma de cualquier rectángulo. Note que si se desea revisar la suma del rectángulo que tiene como esquinas opuestas $[i, j]$ y $[k, l]$ con $i \leq k$ y $j \leq l$, entonces la suma será $a[k, l] - a[k, j] - a[i, l] + a[i, j]$.

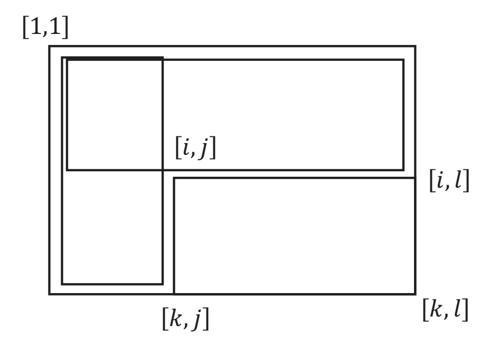



*Figura 6.1* Esquema de la estrategia que se va a usar

## Solución

**Proc**: Máxima Suma
- **Entero:** $n$, $i$, $j$, $a[50, 50]$, $ans$, $k$, $l$, $temp$
- **Lea** $n$
- **Para**$(i = 1, n, 1)$**Haga**
  - **Para**$(j = 1, n, 1)$**Haga**
    - **Lea** $a[i, j]$
    - **Si**$(i > 1)$**Entonces**
      - $a[i, j] \leftarrow a[i, j] + a[i - 1, j]$
    - **F.Si**
    - **Si**$(j > 1)$**Entonces**
      - $a[i, j] \leftarrow a[i, j] + a[i, j - 1]$
    - **F.Si**

1 2 3

```
1 2 3
      Si(i > 1 y j > 1)Entonces
        a[i, j] ← a[i, j] − a[i − 1, j − 1]
      F.Si
    Fin_Para
  Fin_Para
  ans ← −127 * 100 * 100
  Para(i = 1, n, 1)Haga
    Para(j = 1, n, 1)Haga
      Para(k = i, n, 1)Haga
        Para(l = j, n, 1)Haga
          temp ← a[k, l]
          Si(i > 1)Entonces
            temp ← temp − a[i − 1, l]
          F.Si
          Si(j > 1)Entonces
            temp ← temp − a[k, j − 1]
          F.Si
          Si(i > 1 y j > 1)Entonces
            temp ← temp − a[i − 1, j − 1]
          F.Si
          Si(temp > ans)Entonces
            ans ← temp
          F.Si
        Fin_Para
      Fin_Para
    Fin_Para
  Fin_Para
  Escriba ans
Fin_proc
```

# 6.3 Conteo de dígitos

## Enunciado

Diana va a escribir una lista de todos los números enteros positivos entre $A$ y $B$, inclusive, en base 10 y sin ceros a la izquierda. Ella quiere saber cuántas veces se ha escrito cada dígito.

### Entrada

Cada caso de prueba está dado en una línea que contiene dos enteros, $A$ y $B$ $(1 \leq A \leq B \leq 10^8)$. Al último caso de prueba le sigue una línea que contiene dos ceros.

**Salida**

Para cada caso de prueba se debe imprimir una sola línea con 10 enteros, que representan el número de veces que cada dígito se utiliza para escribir todos los números enteros entre $A$ y $B$, inclusive, en base 10, y sin ceros a la izquierda.

**Entrada Ejemplo**

1 9
12 321
5987 6123
12345678 12345679
0 0

**Salida Ejemplo**

0 1 1 1 1 1 1 1 1 1
61 169 163 83 61 61 61 61 61 61
134 58 28 24 23 36 147 24 27 47
0 2 2 2 2 2 2 2 1 1

## Análisis

A manera de ejemplo suponga que se desea contar el número de veces que se usa el dígito 8 al escribir los números del 1 al 1234. Note que en los siguientes números el dígito 8 se está usando en las unidades 1228, 1218, 1208, ..., 8, es decir, el dígito 8 como unidad se está usando 1234/8 veces. Cuando se revisan las decenas, el dígito 8 se utiliza en los números $118x$, $108x$, $98x$, ..., $8x$, de donde se tiene 12 casos. Sin embargo, para cada caso $x$ puede tomar cualquier cifra, por lo tanto, se tiene en total 120 para cuando 8 está en las decenas. Similarmente, si 8 está en las centenas, entonces el único caso sería $8xx$; no obstante, $xx$ puede tomar cualquier valor entre 00 y 99, lo cual implica que se tiene 100 casos más; en total se usó $123+120+100 = 343$ veces el dígito 8.

De igual forma, si se deseara contar el número de veces que se usa el dígito 8 al escribir los números del 1 al 4233434, por ejemplo, entonces se tendría que se usa $423343 + 423340 + 423300 + \cdots + 400000$ veces.

Sin embargo, en los casos anteriores, la cifra buscada tiene un valor mayor que cualquiera de las cifras del número límite. Por ejemplo, suponga que se desea contar el número de veces que se usa el dígito 4 al escribir los números del 1 al 352471.

Y así, cuando el 4 aparece en las unidades se tendría 35247 casos; cuando el dígito 4 aparece en las decenas se tendría que los números $35244x$, $35234x$, $\ldots$, $4x$, es decir, $3524+1$ casos, donde cada caso cuenta por 10, es decir, se tienen 35250 casos más. Cuando el dígito 4 aparece en las centenas, se tiene los casos $3524xx$, $3514xx$, $3504xx$, $\ldots$, $4xx$; cada uno de estos casos, excepto el primero, contribuye por 100; el primero contribuye por "la parte derecha" del número incrementado en 1, es decir, por $71+1=72$. Y de igual modo, se sigue hasta terminar el conteo.

Finalmente, el número de veces que se usa el dígito $d$ al escribir los números entre 1 y $n = \overline{a_k \cdots a_1 a_0}$ es igual a

$$\overline{a_k \cdots a_1} + \overline{a_k \cdots a_2} \cdot 10 + \cdots + \overline{a_k} \cdot 10^{k-1} = \sum_{i=0}^{k-1} \left\lfloor \frac{n}{10^{i+1}} \right\rfloor \cdot 10^i = S$$

Si para algún $i$, $a_i = d$, entonces a $S$ se le debe sumar $\overline{a_{i-1}a_{i-2}\cdots a_0}+1$, y si $a_i > d$, entonces a $S$ se le debe sumar $10^i$.

Note que de lo anterior, para contar el número de veces que aparece se usa un dígito $d$ entre 1 y $n$, hay que hacer tantas iteraciones como el número de cifras que tenga $n$, por lo tanto, este método resulta ser bastante eficiente; su complejidad es $O(\log n)$.

Lo anterior implica que para contar el uso de un dígito $d$ al escribir los números entre $a$ y $b$, inclusive, basta contar entre 1 y $b$ y a esta cantidad restarle el uso del dígito $d$ entre 1 y $a-1$.

## Solución

```
Proc: Conteo de cifras
| Entero: sw, a, b, ans[10], d, i
| sw ← 0
| Mq(sw = 0)Haga
| | Lea a, b
| | Si(a = 0 y b = 0)Entonces
| | | sw ← 1
| | Sino
| | | Para(d = 0, 9, 1)Haga
| | | | ans[d] ← contar(b, d)
| | | | Si(a − 1 > 0)Entonces
| | | | | ans[d] ← ans[d] − contar(a − 1, d)
| | | | F.Si
| | | Fin_Para
| | | Para(i = 0, 9, 1)Haga
| | | |
1 2 3 4
```

```
1 2 3 4
│ │ │ │ Si(i > 0)Entonces
│ │ │ │ │ Escriba " "
│ │ │ │ F.Si
│ │ │ │ Escriba ans[i]
│ │ │ Fin_Para
│ │ F.Si
│ Fin_Mq
Fin_proc
```

```
contar(x, digit)
│ Entero: pot, ans, izq, der, actual
│ pot ← 1, ans ← 0
│ izq ← x, der ← 0
│ Mq(izq > 0)Haga
│ │ actual ← izq mod 10
│ │ izq ← izq/10
│ │ ans ← ans + izq * pot
│ │ Si(actual = digit)Entonces
│ │ │ ans ← ans + der + 1
│ │ Sino Si(actual > digit)
│ │ │ ans ← ans + pot
│ │ F.Si
│ │ pot ← pot * 10
│ │ der ← x mod pot
│ Fin_Mq
│ Si(digit = 0)Entonces
│ │ ans ← ans − (pot − 1)/9
│ F.Si
│ Devuelva ans
Fin_contar
```

## 6.4 Serpientes y escaleras

### Enunciado

Fuente: UVA Online Judge, Problema 647

"Serpientes y Escaleras" es un popular juego de mesa para niños. El tablero tiene casillas que están numeradas del 1 al 100, y los jugadores tienen contadores que inician en la casilla 0. Los jugadores se turnan para tirar un dado con los números del 1 al 6 en él, y cada uno mueve su contador hacia adelante el número de casillas que corresponden al número del dado (la casilla a la que llegan se encuentra sumando el

número del dado y el número de la casilla en la que se encuentran). Gana la primera persona en llegar a la casilla 100.

Lo interesante de este juego se debe al hecho de que hay pares de casillas que se conectan entre sí mediante “escaleras” (que conecta una casilla de número menor a una casilla de número más alto) y “serpientes” (que van de mayor a menor). Si una ficha llega sobre el inicio de una serpiente o una escalera (es decir, esta es la casilla alcanzada después de lanzar el dado), a continuación el contador se mueve a la casilla correspondiente al final de la serpiente o escalera. Tenga en cuenta que llegar a la casilla del final de una escalera o serpiente no tiene ningún efecto, solo cuenta la casilla de origen.

Por otra parte, hay algunas casillas tales que si un jugador cae en ellas, entonces debe perder el próximo turno, o lanzar inmediatamente el dado de nuevo (como si fuese un extraturno), dependiendo de lo que esté escrito en la pizarra. Una casilla “pierde un turno” o “turno extra” nunca está al comienzo o al final de una escalera o serpiente. Si un jugador está en la plaza 95 o superior, entonces cualquier dado que lance e implique superar la casilla 100 será ignorado. Por lo tanto, un jugador en la casilla 99 debe ignorar todos los tiros que no sean 1.

**Entrada**

La entrada comenzará con un conjunto de menos de 1000 tiros que deben utilizarse para todos los juegos; a partir de cada nuevo juego, el primer jugador lanza el primer número de la serie, el siguiente jugador lanza el segundo número, y así sucesivamente. Este conjunto de lanzamientos de dados será simplemente una lista de números aleatorios entre 1 y 6, separados por espacios, con no más de 80 caracteres en cada línea. Se termina con el número 0.

Después del conjunto de lanzamientos de dados se deben leer uno o más juegos. Cada juego tiene tres partes.

La primera parte es una línea que contiene un único número que indica el número de jugadores en el juego. Esto será más de 1 y menos de 6. A continuación se describe las dos partes del tablero.

En la primera parte se enumeran en el tablero las escaleras y las serpientes; cada escalera o serpiente se definen en una sola línea. Cada uno se da por dos números, de 1 a 99, separados por uno o más espacios. El primer número indica la casilla de origen y el segundo la casilla final, por lo que es una escalera si el primer número es menor que el segundo número y una serpiente si el orden es al revés. Las definiciones serpiente/escalera se terminan con una línea que contiene dos 0’s.

En la segunda parte se describen las casillas “pierde un turno/turno extra”, si las hay. Estos son números individuales, una por línea, la definición de las casillas. Si el número es negativo, su negativo representa una casilla “pierde un turno”, y si es positivo representa una casilla “turno extra”. (Por ejemplo, -16 significa que

la casilla 16 en el tablero es un casilla de “pierde un turno”, mientras que un 25 significa que los jugadores que lleguen a la casilla 25 deben lanzar el dado de nuevo inmediatamente).

El final de este conjunto de descripciones, y de la descripción del juego, se da por un único 0. El fin de todas las descripciones de los juegos viene dada por un juego con el número de jugadores igual a 0.

### Salida

La salida debe ser una línea por cada juego en la entrada, dando el número del jugador que gana el juego. Cada partido determinará un ganador en un menor número de tiros que los que se tienen en el inicio de los datos.

### Entrada Ejemplo

```
3 6 3 2 5 1 3 4 2 3 1 2 0
2
6 95
99 1
0 0
-3
98
0
2
3 99
6 90
0 0
0
0
```

### Salida Ejemplo

```
2
2
```

## Análisis

Lo que primero se debe hacer es almacenar la información de los lanzamientos; esto se hará en un arreglo llamado *dados*, de tal manera que se pueda usar en los distintos juegos descritos posteriormente. Luego, para cada juego, se deben leer las escaleras y serpientes y las casillas especiales, y el número de jugadores. Sin embargo,

note que para almacenar las escaleras y serpientes basta con tener un arreglo del tamaño $n \times 2$, en la cual una casilla es el inicio y la otra es el final, es decir, no hace falta distinguir entre si es escalera o serpiente. Este arreglo es *escalera*. Si en algún turno la posición de algún jugador llega a coincidir con alguna de la primera columna del arreglo, entonces su posición cambia automáticamente a la de la segunda columna correspondiente. Con otras palabras, si la casilla en la que se encuentra el jugador es $jug[turno]$, y existe un $ci$ tal que $jug[turno] = escalera[ci, 1]$, entonces a $jug[turno]$ se le asigna $escalera[ci, 2]$.

Un arreglo llamado *jug_turno* determinará si un jugador pierde o no turno. Por lo tanto, note que ganar un turno es equivalente a que el resto pierda un turno. Conforme lo anterior, cuando un jugador llegue a una casilla de turno extra, entonces, en vez de "asignarle" un turno extra, lo que se hará es cambiar el estado de *jug_turno* para todos los jugadores a *verdadero*, excepto para el jugador que ganó turno.

Por último, se asignará el número de jugador, como el residuo módulo el número de jugadores, por lo tanto, el jugador 1 será el jugador 0 (es el primer residuo), el jugador 2 será el jugador 1, y así sucesivamente. Sin embargo, al imprimir el resultado, el jugador 0 no existe, por lo tanto, a la salida se le sumará 1.

## Solución

**Proc**: Escaleras y serpientes
**Entero:** $no_dado$, $dados[50]$, $sw$, $jug_no$, $no_escal$, $escalera[50, 50]$, $especia$
**Entero:** $esp_sqr[50]$, $ci$, $jug[50]$, $sw1$, $time$, $turno$, $sw2$
**Booleano:** $jug_turno[50]$
$no_dado \leftarrow 1$
**Lea** $dados[no_dado]$
**Mq**$(dados[no_dado] \neq 0)$**Haga**
  $no_dado \leftarrow no_dado + 1$
  **Lea** $dados[no_dado]$
**Fin_Mq**
$sw \leftarrow 0$
**Mq**$(sw = 0)$**Haga**
  // Se lee el número de jugadores
  **Lea** $jug_no$
  **Si**$(jug_no > 0)$**Entonces**
    // Se lee las escaleras en el arreglo *escalera*; el primer
    // índice corresponde a la casilla de inicio y el segundo a la
    // casilla de llegada
    $no_escal \leftarrow 1$
    **Lea** $escalera[no_escal, 1], escalera[no_escal, 2]$
1 2 3

```
1 2 3
      Mq(escalera[no_escal, 1] ≠ 0 y escalera[no_escal, 2] ≠ 0)Haga
         no_escal ← no_escal + 1
         Lea escalera[no_escal, 1], escalera[no_escal, 2]
      Fin_Mq
      // Se lee las casillas especiales, las que dan un nuevo turno
      // o con las que se pierde un turno
      especial ← 1
      Lea esp_sqr[especial]
      Mq(esp_sqr[especial] ≠ 0)Haga
         especial ← especial + 1
         Lea esp_sqr[especial]
      Fin_Mq
      // Se inician todos los jugadores en la casilla 0 y sin perder turno
      Para(ci = 1, jug_no, 1)Haga
         jug[ci] ← 0
         jug_turno[ci] ← false
      Fin_Para
      // Se procede a jugar
      sw1 ← 0
      time ← 1
      Mq(sw1 = 0 y time < no_dado)Haga
         time ← time + 1
         turno ← time mod jug_no
         Si(jug_turno[turno] = false)Entonces
            Si(jug[turno] + dados[time] < 100)Entonces
               jug[turno] ← jug[turno] + dados[time]
               sw2 ← 0
               no_escal ← 1
               // Ver si está en el inicio de una escalera o serpiente
               Mq(sw2 = 0 y ci < no_escal)Haga
                  no_escal ← no_escal + 1
                  Si(jug[turno] = escalera[ci, 1])Entonces
                     jug[turno] ← escalera[ci, 2]
                     sw2 ← 1
                  F.Si
               Fin_Mq
               // Ver si está en una casilla especial
               Si(ci > no_escal)Entonces
                  sw2 ← 0
                  ci ← 1
                  Mq(ci < especial y sw2 = 0)Haga
                     ci ← ci + 1
1 2 3 4 5 6 7 8
```

```
1 2 3 4 5 6 7 8
              Si(jug[turno] = abs(esp_sqr[ci]))Entonces
                Si(esp_sqr[ci] ≥ 0)Entonces
                  Para(ck = 1, jug_no, 1)Haga
                    Si(ck ≠ turno)Entonces
                      // Se hará que los otros jugadores pierdan un turno
                      jug_turno[ck] ← verdadero
                    F.Si
                  Fin_Para
                Sino
                  jug_turno[turno] ← verdadero
                F.Si
              F.Si
            Fin_Mq
          F.Si
        Sino Si(jug[turno] + dados[time] = 100)
          sw1 ← 1
        F.Si
      Sino
        jug_turno[turno] ← falso
      F.Si
    Fin_Mq
    // Finalmente se escribe el jugador ganador
    Escriba turn + 1
  Sino
    sw ← 1
  F.Si
Fin_Mq
Fin_proc
```

# 6.5 Estaciones de gas

## Enunciado

Fuente: UVA Online Judge, Problema 12321

$G$ estaciones de gas están autorizadas para operar en una carretera de longitud $L$. Cada estación es capaz de vender combustible sobre una zona específica de influencia, que se define como el intervalo cerrado $[x - r, x + r]$, donde $x$ es la ubicación de la estación en el carretera ($0 \leq x \leq L$) y $r$ es el radio de cobertura ($0 < r \leq L$). Los puntos sobre la carretera cubiertos por una estación de gas son los que están dentro de su radio de cobertura.

Es claro que las áreas de influencia pueden interferir, causando disputas entre las estaciones de servicio correspondientes. Parece que es mejor cerrar algunas estaciones, tratando de minimizar estas interferencias, sin reducir la disponibilidad del servicio a lo largo del camino.

Los propietarios se han comprometido a cerrar algunas estaciones de servicio con el fin de evitar el mayor número posible de las disputas. Usted ha sido contratado para escribir un programa para determinar el número máximo de estaciones de servicio que pueden ser cerradas para que cada punto de la carretera se encuentre en el área de influencia de alguna estación no cerrada. Por cierto, si algún punto sobre el camino no está cubierto por ninguna estación de servicio, usted debe reconocer la situación e informar al respecto.

## Entrada

La entrada consta de varios casos. La primera línea de cada caso consiste de dos enteros, $L$ y $G$ (separados por un espacio), que representan la longitud del camino y el número de estaciones de gas, respectivamente ($1 \leq L \leq 10^8$, $1 \leq G \leq 10^4$). Cada una de las siguientes $G$ líneas consiste de dos enteros, $x_i$ y $r_i$, (separados por un espacio), siendo $x_i$ la ubicación y $r_i$ el radio de cobertura de la $i$-ésima estación de gas ($0 \leq x_i \leq L$, $0 < r_i \leq L$). El último caso de prueba está seguido por una línea que contiene dos ceros.

## Salida

Para cada caso de prueba se debe imprimir una línea con el número máximo de estaciones de gas que puedan ser eliminadas, de manera que cada punto en la carretera pertenece a la zona de influencia de alguna estación no cerrada. Si algún punto en la carretera no está cubierto por ninguna de las estaciones iniciales de gas $G$, imprimir '-1' como la respuesta para tal caso.

## Entrada Ejemplo

```
40 3
5 5
20 10
40 10
40 5
5 5
11 8
20 10
30 3
40 10
```

40 5
0 10
10 10
20 10
30 10
40 10
40 3
10 10
18 10
25 10
40 3
10 10
18 10
25 15
0 0

**Salida Ejemplo**

0
2
3
-1
1

## Análisis

Note que en la entrada específica $G$ puede tomar valores de hasta $10^4$. Lo anterior implica que si se desea revisar todas las posibles combinaciones y ver cuál de ellas tiene el menor número de estaciones de gasolina requeridas para cubrir toda la carretera, se necesitan $2^{10^4} = 2^{10000}$ iteraciones, lo cual toma demasiado tiempo para un computador.

Lo primero que se debe hacer es almacenar la información de los dos extremos de cada estación de gas; esto se se debe hacer en el arreglo $gs$, el cual es de tamaño $n \times 2$, siendo $n$ el número de estación de gas; no obstante, de la entrada se obtiene la posición y el radio de cobertura, por lo tanto, es posible que $x - r < 0$ y $x + r > L$, de donde, para cada estación de gas que se lee se debe escoger el máximo valor entre 0 y $x - r$ y el mínimo valor entre $L$ y $x + r$.

A su vez, en lugar de contar el número máximo de estaciones que se pueden eliminar, se debe contar el número mínimo de estaciones necesarias para cubrir toda la carretera. Suponga que se tiene que una estación $i$ que no se va a cerrar y su punto de inicio es $gs[i, 1]$ y su punto de fin es $gs[i, 2]$. La siguiente estación $j$ que se ha de seleccionar debe cumplir que $gs[k, 1] \leq gs[i, 2]$ y que $gs[j, 2]$ tiene el mayor

valor posible. Note que de esta forma se minimiza el número de estaciones de gas necesarias. Por último, es posible ordenar las estaciones de gas de tal forma que se facilite la búsqueda de $j$ una vez se tiene $i$.

Si se dice que una estación de gas $i$ es menor que otra estación $j$ si $gs[i,1] < gs[j,1]$ en el caso de que $gs[i,1] \neq gs[j,1]$ o si $gs[i,2] > gs[j,2]$ en caso contrario. De esta forma, si se ordena mediante un método de ordenamiento el arreglo $gs$, la primera estación de gas debe cumplir que $gs[1,1] = 0$, si no se debe imprimir un '-1'. Y para esta primera estación se inicia la búsqueda de la segunda estación de gas.

Según la estrategia mencionada, si se encuentra en una estación gas $i$ y se desea buscar la siguiente, un ciclo Para interno empezará la búsqueda desde $j = i$ hasta $j = n$, y en la variable *mejor* se almacenará la estación que tiene su límite de cobertura $x + r$ máximo, este se declara en $i$ y la idea es encontrar un $j > i$ que cumpla las condiciones mencionadas $gs[j,1] \leq gs[i,2]$ y $gs[j,2] > gs[mejor,2]$. En caso de que el valor de la variable *mejor* sea $i$ luego de terminado el ciclo Para, quiere decir que no se encontró otra estación. En este caso, el ciclo Mientras que externo termina.

A su vez, una vez terminado el ciclo, la variable *cubierto* determinará desde 0 hasta qué punto de la carretera se ha cubierto. Por lo tanto, cada vez que se encuentra una nueva estación a la "derecha" de la estación $i$, la variable *cubierto* se actualiza con el valor de $gs[mejor,2]$.

Una vez terminado el ciclo Mientras que externo, se verifica si $cubierto = L$ o no. En caso de serlo, se imprime $n - ans$, y en caso contrario, '-1'.

## Solución

**Proc**: Estaciones de Gas
| **Entero:** $sw$, $L$, $n$, $i$, $x$, $r$, $gs[50,50]$, $cubierto$, $ans$, $sw2$, $mejor$, $j$
| $sw \leftarrow 0$
| **Mq**$(sw = 0)$**Haga**
| | **Lea** $L, n$
| | **Si**$(L \neq 0 \text{ o } n \neq 0)$**Entonces**
| | | **Para**$(i = 1, n, 1)$**Haga**
| | | | **Lea** $x, r$
| | | | $gs[i,1] \leftarrow \max(0, x - r)$
| | | | $gs[i,2] \leftarrow \min(L, x + r)$
| | | **Fin_Para**
| | | Ordenar($gs$, $n$)
| | |
| | | **Si**$(gs[1,1] > 0)$**Entonces**
| | | | **Escriba** $-1$
1 2 3 4

```
1 2 3 4
│ │ │ Sino
│ │ │ │ cubierto ← 0
│ │ │ │ ans ← 0, i ← 1
│ │ │ │ sw2 ← 0
│ │ │ │
│ │ │ │ Mq(i ≤ n y sw2 = 0)Haga
│ │ │ │ │ ans ← ans + 1
│ │ │ │ │ mejor ← i
│ │ │ │ │ Para(j = i, n, 1)Haga
│ │ │ │ │ │ Si(gs[j, 1] ≤ gs[i, 2])Entonces
│ │ │ │ │ │ │ Si(gs[j, 2] > gs[mejor, 2])Entonces
│ │ │ │ │ │ │ │ mejor ← j
│ │ │ │ │ │ │ F.Si
│ │ │ │ │ │ F.Si
│ │ │ │ │ Fin_Para
│ │ │ │ │
│ │ │ │ │ cubierto ← gs[mejor, 2]
│ │ │ │ │
│ │ │ │ │ Si(mejor = i)Entonces
│ │ │ │ │ │ sw2 ← 1
│ │ │ │ │ Sino
│ │ │ │ │ │ i ← mejor
│ │ │ │ │ F.Si
│ │ │ │ Fin_Mq
│ │ │ │
│ │ │ │ Si(cubierto < L)Entonces
│ │ │ │ │ Escriba −1
│ │ │ │ Sino
│ │ │ │ │ Escriba n − ans
│ │ │ │ F.Si
│ │ │ F.Si
│ │ Sino
│ │ │ sw ← 1
│ │ F.Si
│ Fin_Mq
Fin_proc

Ordenar(gs, n)
│ Entero: i, j, aux
│ Para(i = 1, n − 1, 1)Haga
│ │ Para(j = i + 1, n, 1)Haga
│ │ │ Si(gs[i, 1] > gs[j, 1] o gs[i, 2] < gs[j, 2])Entonces
│ │ │ │ aux ← gs[i, 1]
│ │ │ │ gs[i, 1] ← gs[j, 1]
1 2 3 4
```

```
1 2 3 4
│ │ │ │ gs[j,1] ← aux
│ │ │ │
│ │ │ │ aux ← gs[i,2]
│ │ │ │ gs[i,2] ← gs[j,2]
│ │ │ │ gs[j,2] ← aux
│ │ │ F.Si
│ │ Fin_Para
│ Fin_Para
Fin_Ordenar
```

# 6.6 Radares de inspección

## Enunciado

Fuente: UVA Online Judge, Problema 12323

Radares Inc. es un fabricante de radares reconocido en todo el mundo, cuya reputación yace en sus estrictos procedimientos de control de calidad y una gran variedad de modelos de radares que se ajustan a todos los presupuestos. Esta compañía lo ha contratado a usted para desarrollar una inspección detallada, que consiste en una secuencia de experimentos electrónicos en un modelo específico de vigilancia.

Hay un campo representado por un plano en coordenadas polares que contiene $N$ objetos colocados en posiciones con coordenadas polares enteras. El modelo de inspección se encuentra en el origen $(0, 0)$ del campo y puede detectar objetos a una distancia inferior a su rango de detección $R$ a través de un área de escaneado definida por cuatro parámetros de ajuste $\alpha$, $A$, $h$, y $H$, cuyo significado se ilustra en la figura 6.2.

Formalmente, el área de escaneado del modelo es la región descrita por el conjunto de puntos polares

$$\{(r, \theta) \mid h \leq r < h + H,\ \alpha \leq \theta \leq \alpha + A\}$$

$\alpha$, $A$, $h$ y $H$ son cuatro valores enteros, donde

- $\alpha$ especifica el ángulo de inicio del área de escaneo del radar $(0 \leq \alpha < 360)$,
- $A$ especifica el ángulo de apertura del área de escaneo del radar $(0 \leq A < 360)$,
- $h$ determina con el radio interno del área de escaneo del radar $(0 \leq h < R)$, y
- $H$ determina la altura del área de escaneo del radar $(0 \leq H \leq R)$.

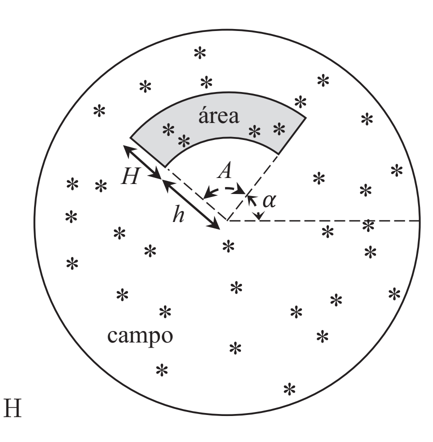



*Figura 6.2* Parámetros de un radar de inspección

Un objeto ubicado en $(r, \theta)$ se mostrará en el modelo si $h \leq r < h + H$ y $\alpha \leq \theta \leq \alpha + A$, donde la última desigualdad debe ser entendida en módulo 360° (es decir, la suma y la comparación de los ángulos en un círculo; por ejemplo, 120° y 480° son el mismo ángulo).

Dado $N$ objetos ubicados en el campo, usted debe desarrollar una inspección del modelo de vigilancia a través de la implementación de $E$ experimentos con parametrizaciones específicas. Para cada experimento tiene que encontrar el número máximo de objetos en el campo que el radar debe mostrar si los parámetros $\alpha$ $(0 \leq \alpha < 360)$ y $h$ $(0 \leq h < R)$ son libres de elegir (como números enteros) y los parámetros $H$ $(1 \leq H \leq R)$ y $A$ $(0 \leq A < 360)$ están dados.

## Entrada

La entrada consta de varios casos. Cada caso de prueba está descrito como se muestra a continuación:

- Una línea con dos números enteros, $N$ y $R$, separados por un espacio en blanco, representando (respectivamente) el número de objetos localizados en el campo y rango de detección del modelo $(1 \leq N \leq 10^4$, $2 \leq R \leq 10^2)$.
- Cada una de las siguientes $N$ líneas contiene dos números enteros, $r_i$ y $\theta_i$, separados por un espacio en blanco, que especifican las coordenadas polares enteras $(r_i, \theta_i)$ del $i$-ésimo objeto $(1 \leq r_i < R$, $0 \leq \theta_i < 360$, $1 \leq i \leq N)$.
- La siguiente línea tiene un número entero, $E$, que indica el número de experimentos de la inspección $(1 \leq E \leq 10^2)$.
- Cada uno de las siguientes $E$ líneas tiene dos números enteros, $H_j$ y $A_j$, separados por un espacio en blanco, que representan (respectivamente) la altu-

ra y el ángulo de apertura fijados que parametrizan el $j$-ésimo experimento ($1 \leq H_j \leq R$, $0 \leq A_j < 360$, $1 \leq j \leq E$).

Para cada caso de prueba se puede suponer que no hay dos objetos diferentes situados en la misma coordenada polar. El último caso de prueba es seguido por una línea que contiene dos ceros.

**Salida**

Para cada caso de prueba de la entrada se deben imprimir $E$ líneas, de las cuales la $j$-ésima línea contiene el número máximo de objetos en el campo que el radar puede mostrar de acuerdo con la parametrización dada para el $j$-ésimo experimento ($1 \leq j \leq E$).

**Entrada Ejemplo**

```
6 100
15 7
15 60
40 15
50 15
45 30
45 90
2
2 1
100 359
9 100
15 7
15 60
40 15
50 15
45 30
45 90
40 45
50 45
78 100
6
100 359
11 30
10 30
11 29
5 30
11 10
0 0
```

**Salida Ejemplo**

1
6
9
5
3
3
2
2

## Análisis

La matriz $mat$ contendrá la ubicación de todos los objetos sobre el campo. Puesto que se debe trabajar con los ángulos módulo 360°, para cada objeto con coordenadas $(r_i, \theta_i)$ que se lea se colocará un 1 en $mat[r_i, \theta_i]$ y en $mat[r_i, \theta_i + 360]$, de esta forma se puede trabajar los ángulos módulo 360°. Lo anterior se debe a que cualquier región en la segunda coordenada tendrá todos sus puntos entre $\alpha$ y $\alpha + A$. El valor mínimo que puede tomar $\alpha$ es 0, y el valor máximo que puede tomar $\alpha + A$ es $360 + 360 = 720$, por lo tanto, duplicando la región es posible trabajar el problema, evitando problemas con los ángulos módulo 360°. La utilidad de esta estrategia se verá a continuación.

La función 'encontrarmax' determinará el número máximo de objetos encerrados por una región. Para ello, se escoge una esquina de la región $[i, j]$ entre todos los posibles valores, es decir, $i \in [1, R - H]$ y $j \in [1, 360]$. La otra esquina está determinada por $[i + H, j + A]$, donde $H$ y $A$ son las alturas y el ángulo de apertura del área de escaneo del radar, respectivamente. De esta forma, se está barriendo todas las posibles ubicaciones de dicha área.

Para determinar rápidamente el número de puntos que están sobre un área se debe hacer un precálculo similar al que se hizo en el problema de Máximas sumas. En este caso, el valor en $sum[i, j]$ significa el número de objetos localizados en la región encerrada por el rectángulo de esquinas $[1, 1]$ y $[i, j]$. Una vez hecho esto, la función 'rectang' calcula el número de objetos localizados en la región encerrada.

Finalmente, un ciclo Para lee los $E$ experimentos, llama a la función 'encontrarmax' e imprime el número máximo de objetos que se pueden encontrar en la región con los parámetros ingresados.

## Solución

```
Proc: Radares
  Entero: MAXR, MAXW, sw, N, R, i, j, mat[100,720], r, tht
  Entero: sum[100,720], E, dist, ang, ans
  MAXR ← 100
  MAXW ← 720
  sw ← 0
  Mq(sw = 0)Haga
    Lea N, R
    Si(N ≠ 0 o R ≠ 0)Entonces
      // Se inicia toda la matriz mat en 0
      Para(i = 1, R, 1)Haga
        Para(j = 1, 720, 1)Haga
          mat[i, j] ← 0
        Fin_Para
      Fin_Para

      Para(i = 1, N, 1)Haga
        Lea r, tht
        mat[r, tht] ← 1
        mat[r, tht + 360] ← 1
      Fin_Para

      // Pre-computo
      Para(i = 1, MAXR, 1)Haga
        Para(j = 1, MAXW, 1)Haga
          sum[i, j] ← mat[i, j]
          Si(i > 0)Entonces
            sum[i, j] ← sum[i, j] + sum[i − 1, j]
          F.Si
          Si(j > 0)Entonces
            sum[i, j] ← sum[i, j] + sum[i, j − 1]
          F.Si
          Si(i > 0 y j > 0)Entonces
            sum[i, j] ← sum[i, j] − sum[i − 1, j − 1]
          F.Si
        Fin_Para
      Fin_Para

      Lea E
      Para(i = 1, E, 1)Haga
        Lea dist, ang
1 2 3 4
```

```
1 2 3 4
         ans ← encontrarmax(dist,ang,R,sum)
         Escriba ans
      Fin_Para
    Sino
      sw ← 1
    F.Si
  Fin_Mq
Fin_proc

encontrarmax(H, A, R, sum)
  Entero: ans, i, j, op
  ans ← 0
  Para(i = 1, R − H, 1)Haga
    Para(j = 1, 360, 1)Haga
      op ← rectang(i, j, i + H, j + A, sum)
      Si(op > ans)Entonces
        ans ← op
      F.Si
    Fin_Para
  Fin_Para
  Devuelva ans
Fin_encontrarmax

rectang(r1, c1, r2, c2, sum)
  Entero: ans
  ans ← sum[r2, c2]
  Si(r1 > 0)Entonces
    ans ← ans − sum[r1 − 1, c2]
  F.Si
  Si(c1 > 0)Entonces
    ans ← ans − sum[r2, c1 − 1]
  F.Si
  Si(r1 > 0 y c1 > 0)Entonces
    ans ← ans + sum[r1 − 1, c1 − 1]
  F.Si
  Devuelva ans
Fin_rectang
```

## 6.7 Estacionamientos

### Enunciado

Fuente: UVA Online Judge, Problema 603

En una universidad, un pequeño aparcamiento está dispuesto en una forma rectangular (ver figura 6.3), con 20 espacios numerados 1, 2, 3 . . ., 19, 20. El flujo de tráfico es en sentido contrario a las agujas del reloj.

Note que la primera posición que se encuentra al entrar es 1 y la última es 20. Los coches pueden salir o continuar conduciendo en sentido contrario a las agujas del reloj. Los siguientes supuestos se aplican a este problema:

- Al principio el parqueadero está lleno (todos los 20 espacios están ocupados por coches).
- Además de los 20 carros que ya están parqueados en el lote, $K$ autos están esperando un espacio de parqueo disponible ($1 \leq K \leq 20$).
- Cada auto que espera se coloca detrás de uno de los espacios ocupados. Cuando se desocupa una posición, el espacio es ocupado ya sea por el coche de espera en esa posición o, si no hay coche a la espera en esa posición, por el coche más cercano, teniendo en cuenta que el flujo de tráfico es en un solo sentido. (Hay espacio suficiente en cada posición para que el coche aparcado en esa posición pueda salir y el coche que está esperando en esa posición luego se estacione.)
- Cuando un auto avanza $N$ posiciones a un lugar libre, el resto de autos también avanza $N$ posiciones. Dado que el lote es circular, avanzar 4 posiciones desde la posición 18 significa avanzar a la posición 2.
- Ninguno de los coches en espera sale.

```
          _______________________________________________________
          |                                                      |
          |  20    19    18    17    16    15    14    13    12    11      |
 Salida <<<<       <     <     <     <     <     <     <     <     <       |
          |  v                                                   ^       |
 Entrada  >>>>     >     >     >     >     >     >     >     >     >       |
          |  1     2     3     4     5     6     7     8     9     10      |
          |______________________________________________________|
```

*Figura 6.3* Parqueadero

### Entrada

Escriba un programa que lea la información de entrada estándar. La entrada consiste en una línea que indica el número de conjuntos de datos, una línea en blanco, y los conjuntos de datos separados por una línea en blanco. Cada conjunto de datos se divide en dos partes. La primera parte consta de los números enteros, uno por línea, comenzando en la columna 1, en representación de las posiciones iniciales de autos esperando. Un entero 99 señala el final de esta parte de los datos. La segunda parte consta de los números enteros, en el mismo formato, que representa las posiciones que no se encuentran ocupadas. Similarmente, un entero 99 señala el final de esta parte de los datos.

Las posiciones se liberan a partir del orden en que sus números aparecen en la segunda parte de los datos.

### Salida

La salida de cada conjunto de datos debe consistir en una serie de líneas que dan, para cada posición inicial (espera) del coche, la posición inicial y la posición final del auto con base en la descripción y los supuestos hechos. Las líneas de salida deben aparecer en el mismo orden que el orden de las posiciones iniciales dadas en la entrada. Imprima una línea en blanco entre los conjuntos de datos.

### Entrada Ejemplo

```
1

6
19
17
13
1
99
1
3
20
16
99
```

### Salida Ejemplo

```
El auto de posición inicial 6 se parqueó en 16.
El auto de posición inicial 19 se parqueó en 3.
```

El auto de posición inicial 17 no se parqueó.
El auto de posición inicial 13 se parqueó en 20.
El auto de posición inicial 1 se parqueó en 1.

## Análisis

En un vector llamado *carroID* se almacenará la posición inicial de cada auto, en otro, *carro_donde*, se almacenerá la posición final donde se parquea el auto, en caso de hacerlo, y por último, el vector *carro_parked* indicará si un auto se parqueó o no. Cada espacio de parqueo que se vuelve disponible se lee temporalmente en la variable *salio*. Para determinar qué auto se debe parquear allí, se selecciona el auto más cercano a dicha posición, midiendo su distancia con respecto a dicho auto.

Por lo tanto, si *salio* es donde el auto se debe parquear y $carro_donde[i]$ es la posición de un auto $i$, entonces la distancia de ese auto con respecto al espacio de parqueo es

$$d = \begin{cases} salio - carro_donde[i], & \text{si } salio \geq carro_donde[i] \\ (salio + 20) - carro_donde[i], & \text{si } salio < carro_donde[i] \end{cases}$$

Por ejemplo, si un auto está en la posición 20 y el espacio libre está en la posición 2, note que su distancia es 2 ($(2+20)-20 = 2$); por el contrario, si un auto está en la posición 2 y el espacio libre está en la posición 20, entonces su distancia es $20-2 = 18$.

Por último, se debe encontrar el valor de $i$ tal que $d$ sea el mínimo posible; ese valor de $i$ se almacena en la variable *posmascercana*. En caso de no encontrarse, entonces el $i$-ésimo carro no se parqueó.

## Solución

**Proc**: Estacionamiento
  **Entero:** *casos*, *i*, *sw*, *cont*, *id*, $carroID[50]$, $carro_donde[50]$, *salio*
  **Entero:** *posmascercana*, *mindist*, *d*
  **Booleano:** *primerCaso*, *carro_parked*
  **Lea** *casos*
  $primerCaso \leftarrow verdadero$
  **Para**$(i = 1, casos, 1)$**Haga**
    **Si**$(primerCaso = falso)$**Entonces**
      **Escriba** “ ”
    **F.Si**
    $primerCaso \leftarrow falso$
    $sw \leftarrow 0$
1 2

```
1 2
    cont ← 0
    Mq(sw = 0)Haga
        Lea id
        Si(id ≠ 99)Entonces
            cont ← cont + 1
            carroID[cont] ← id
            carro_donde[cont] ← id
            carro_parked[cont] ← falso
        Sino
            sw ← 1
        F.Si
    Fin_Mq
    sw ← 0
    Mq(sw = 0)Haga
        Lea salio
        Si(salio ≠ 99)Entonces
            posmascercana ← 9999
            mindist ← 9999
            Para(i = 1, cont, 1)Haga
                Si(carro_parked[i] = falso)Entonces
                    Si(carro_donde[i] ≤ salio)Entonces
                        d ← salio − carro_donde[i]
                    Sino
                        d ← (salio + 20) − carro_donde[i]
                    F.Si
                    Si(d < mindist)Entonces
                        mindist ← d
                        posmascercana ← i
                    F.Si
                F.Si
            Fin_Para
            Si(posmascercana = 9999)Entonces
                sw ← 1
            Sino
                carro_parked[posmascercana] ← verdadero
                carro_donde[posmascercana] ← salio
                Para(i = 1, cont, 1)Haga
                    Si(carro_parked[i] = falso)Entonces
                        d ← carro_donde[i] + mindist
                        d ← d mod 20
                        Si(d = 0)Entonces
                            d ← 20
                        F.Si
1 2 3 4 5 6 7
```

```
1 2 3 4 5 6 7
            carro_donde[i] ← d
          F.Si
        Fin_Para
      F.Si
    Sino
      sw ← 1
    F.Si
  Fin_Mq
  Para(i = 1, cont, 1)Haga
    Escriba "El auto de posición inicial ", carroID[i], " "
    Si(carro_parked[i] = verdadero)Entonces
      Escriba "se parqueó en ", carro_donde[i]
    Sino
      Escriba "no se parqueó"
    F.Si
  Fin_Para
 Fin_Mq
Fin_proc
```

# 6.8 Control de vuelo

## Enunciado

Fuente: UVA Online Judge, Problema 12320

Los controladores aéreos agilizan y mantienen un flujo seguro y ordenado del tránsito aéreo en el sistema de control del tráfico aéreo mundial. La profesión de controlador de tránsito aéreo requiere conocimientos muy especializados. Dado que los controladores tienen una responsabilidad muy grande, esta profesión es, según Wikipedia, "considerada en todo el mundo como uno de los trabajos más difíciles hoy en día, y puede ser tremendamente estresante dependiendo de muchas variables (equipos, configuraciones, tiempo, volumen de tráfico, factores humanos, etc)".

Un controlador de tránsito aéreo tiene acceso a la siguiente información de cada avión en la pantalla:

- tamaño: un número entero positivo $r$ indica el radio (medido en metros) de una esfera cuyo centro de seguridad siempre es la posición actual de la aeronave;
- velocidad: un número entero positivo $s$ que indica la velocidad constante (medido en metros por segundo) de la aeronave a lo largo de su recorrido;

- ruta: una secuencia de puntos $(x_1, y_1, z_1)$, $(x_2, y_2, z_2)$, ..., $(x_k, y_k, z_k)$ con coordenadas enteras (medido en metros) en el plano cartesiano tridimensional, lo que indica la ruta seguida por la aeronave antes de que vuelva al inicio de la pista que se encuentra en el punto $(0, 0, 0)$.

Cada aeronave inicia su recorrido en la posición $(x_1, y_1, z_1)$ y, a continuación, vuela directamente en línea recta a una velocidad constante de $(x_1, y_1, z_1)$ a $(x_2, y_2, z_2)$, ..., de $(x_{k-1}, y_{k-1}, z_{k-1})$ a $(x_k, y_k, z_k)$, y, finalmente, a partir de $(x_k, y_k, z_k)$ a $(0, 0, 0)$, donde cada posición es relativa al inicio de la pista. Después de que el avión llega al inicio de la pista desaparece de la pantalla del controlador.

En el día a día, los controladores aéreos se enfrentan a la detección y resolución de conflictos. Por lo tanto, para un controlador de tránsito aéreo es muy importante contar con un sistema de alarma para indicar los puntos específicos donde se debe tomar acciones correctivas para evitar accidentes.

Es de destacar que, geométricamente hablando, se produce una advertencia de conflicto cuando las esferas de seguridad de dos aviones se tocan. Formalmente, una advertencia de conflicto comienza cuando dos aeronaves se aproximan a una distancia inferior o igual a la suma de los radios de sus esferas de seguridad; se mantiene mientras que se cumple esta condición, y termina cuando sus esferas de seguridad se dejan de tocar (es decir, cuando la distancia entre ambos es mayor que la suma de los radios de sus esferas de seguridad). Las distancias se miden con un umbral de error aceptable $\varepsilon > 0$.

A pesar de años de esfuerzos y de los miles de millones de dólares que se han gastado en software informático diseñado para ayudar a controlar el tráfico aéreo, el éxito ha sido limitado en gran medida a la mejora de las herramientas a disposición de los controladores. Sin embargo, hoy usted tiene la oportunidad de mejorar el impacto de los programas informáticos en el mundo en el control del tráfico aéreo.

Usted ha sido contratado por el Centro Internacional de Control de Planificación (ICPC) para determinar la calidad de las rutas de tráfico definidas por el Controlador de Administración de Aeronaves (ACM) a través de la medición de la cantidad de situaciones de peligro que debe arreglar el controlador de tránsito aéreo. Su tarea es escribir un programa que, dada la información de dos aviones, determine el número de diferentes advertencias de conflicto que se produciría si ambas aeronaves siguen la ruta programada de partida al mismo tiempo y acabando en el inicio de la pista.

### Entrada

La primera línea de la entrada contiene el número de casos de prueba. Cada caso de prueba especifica la información de las dos aeronaves estudiadas, en la que cada aeronave se describe como sigue:

- La primera línea contiene tres números enteros, $r$, $s$ y $k$, separados por espacios

en blanco ($1 \leq r \leq 100$, $1 \leq s \leq 1000$, $1 \leq k \leq 100$), donde $r$ es el radio de la esfera de seguridad (en metros), $s$ es la velocidad (en metros por segundo) y $k$ es el número de puntos que definen la ruta de la aeronave.

- Cada una de las siguientes $k$ líneas contiene tres números enteros, $x_i$, $y_i$ y $z_i$, separados por espacios en blanco ($-10^4 \leq x_i \leq 10^4$, $-10^4 \leq y_i \leq 10^4$, $0 \leq z_i \leq 10^4$), que describen las coordenadas (en metros) del punto $i$-ésimo en la ruta de la aeronave ($1 \leq i \leq k$).

Para cada ruta usted puede asumir que $x_i \neq x_{i+1}$, $y_i \neq y_{i+1}$ ó $z_i \neq z_{i+1}$ para todo $1 \leq i \leq k$, y que o $x_k \neq 0$, $y_k \neq 0$ o $z_k \neq 0$. El error aceptable de aproximación es $\varepsilon = 10^{-10}$ m.

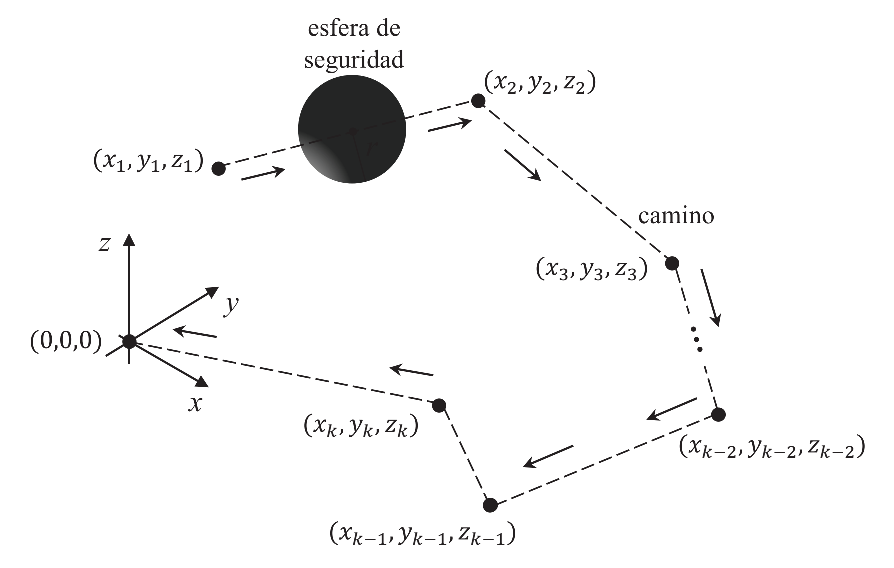



*Figura 6.4* Control de vuelo

**Salida**

Para cada prueba imprima una línea que informe el número de diferentes advertencias de conflicto.

**Entrada Ejemplo**

7
20 300 2
10000 1000 5000
1000 100 500
20 100 3

10000 1000 2000
1000 100 500
100 0 0
20 300 2
0 10000 5000
0 -10000 5000
20 300 3
10000 0 5010
-10000 0 5010
-8000 1000 3010
20 300 2
0 10000 5000
0 -10000 5000
20 300 2
10000 0 5010
-10000 0 5010
25 200 2
3000 6000 3000
4000 5000 3000
25 200 2
3000 6000 3005
4000 5000 3005
20 300 2
5000 4000 3000
4000 5000 3000
20 300 2
3000 6000 3005
4000 5000 3005
10 100 1
-1000 0 0
10 100 2
1000 21 0
-1000 21 0
10 100 1
-1000 0 0
10 100 2
1000 20 0
-1000 20 0

**Salida Ejemplo**

0
1
2
1

1
0
1

## Análisis

La distancia entre dos puntos, $P = (x_1, y_1, z_1)$ y $Q = (x_2, y_2, z_2)$, está dada por

$$d(P,Q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)2}$$

Por lo tanto, es posible calcular el instante del tiempo $t$ en el que se encuentra cada avión al pasar por cada punto de su trayectoria. En el arreglo $A$, de tamaño $(kA+1) \times 4$, y en el arreglo $B$, de tamaño $(kB+1) \times 4$, se guardará la información de cada punto de la trayectoria y el instante de tiempo en el que se encuentra el avión al pasar por dicho punto.

Para calcular el tiempo en el punto $i$ se emplea

$$t_i = \sum_{k=1}^{i} \frac{d(P_k, P_{k+1})}{s} = t_{i-1} + \frac{d(P_i, P_{i-1})}{s}$$

donde $s$ es la velocidad del avión. La varible que servirá de acumulador para esta suma será *sumt*.

Una vez hecho el cálculo anterior es posible determinar funciones que describan la posición de un avión en cualquier instante de tiempo; para ello note que si tiene dos puntos y dos instantes de tiempo correspondientes, $(t_i, \mathbf{r}_i)$ y $(t_{i+1}, \mathbf{r}_{i+1})$, entonces la función que describe la posición del avión para $t_i \leq t \leq t_{i+1}$ es

$$\mathbf{f}(t) = \mathbf{r}_i + \frac{\mathbf{r}_{i+1} - \mathbf{r}_i}{t_{i+1} - t_i}(t - t_i)$$

En función de esto, entonces es posible calcular la distancia entre los dos aviones:

$$\begin{aligned}
d(t) &= \|\mathbf{f}^A(t) - \mathbf{f}^B(t)\| \\
&= \left\| \left( \mathbf{r}_i^A + \frac{\mathbf{r}_{i+1}^A - \mathbf{r}_i^A}{t_{i+1}^A - t_i^A}(t - t_i^A) \right) - \left( \mathbf{r}_i^B + \frac{\mathbf{r}_{i+1}^B - \mathbf{r}_i^B}{t_{i+1}^B - t_i^B}(t - t_i^B) \right) \right\| \\
d^2(t) &= \sum_{x,y,z} \left( x_i^A - x_i^B + \underbrace{\frac{x_{i+1}^A - x_i^A}{t_{i+1}^A - t_i^A}}_{D_{x,A}}(t - t_i^A) - \underbrace{\frac{x_{i+1}^B - x_i^B}{t_{i+1}^B - t_i^B}}_{D_{x,B}}(t - t_i^B) \right)^2 \\
\frac{\partial}{\partial t}(d^2(t)) &= \frac{\partial}{\partial t} \sum_{x,y,z} \left[ \underbrace{x_i^A - x_i^B - \left( D_{x,A} t_i^A - D_{x,B} t_i^B \right)}_{C_x} + \underbrace{\left( D_{x,A} - D_{x,B} \right)}_{M_x} t \right]^2
\end{aligned}$$

$$
\begin{aligned}
2d(t)\frac{\partial d}{\partial t} &= 2\sum_{x,y,z}\left(C_x + M_x t\right) M_x \\
\frac{\partial d}{\partial t} &= \frac{\sum_{x,y,z}\left(C_x + M_x t\right) M_x}{d(t)} = 0
\end{aligned}
$$

de donde

$$\sum_{x,y,z}(M_x^2 t + M_x C_x) = 0.$$

Por lo tanto,

$$t_{\min} = -\frac{\sum_{x,y,z} M_x C_x}{\sum_{x,y,z} M_x^2} = -\frac{M_x C_x + M_y C_y + M_z C_z}{M_x^2 + M_y^2 + M_z^2}$$

Note que $d^2(t)$ es un polinomio cuadrático y que $\sqrt{x} : \mathbb{R}^+ \to \mathbb{R}^+$ es una función biyectiva, por lo tanto, $d(t) = \sqrt{d^2(t)}$ tiene un mínimo en un intervalo $[a, b]$, en $a$, $b$ o en un punto en el interior del intervalo para el cual $\frac{\partial d}{\partial t} = 0$. Por lo tanto, hay que calcular $t_{\min}$ y verificar si $t_{\min}$ está en el intervalo que se está revisando; si no lo está, el mínimo está en uno de los extremos.

Una vez se determina la distancia mínima, se verifica si es mayor o no a la suma de los radios de las esferas de seguridad de cada uno de los aviones. El código simula a continuación el vuelo de los dos aviones, y va acumulando en *cw* el número de conflictos posibles que se pueden llegar a presentar.

## Solución

**Proc**: Control de vuelo
- **Real:** $e$, $sumt$, $x$, $y$, $z$, $A[50, 50]$, $d$, $B[50, 50]$, $t1$, $t2$, $DsxA$, $DsxB$, $Cx$, $Mx$
- **Real:** $DsyA$, $DsyB$, $Cy$, $My$, $DszA$, $DszB$, $Cz$, $Mz$, $tmin$, $dmin1$, $dmin2$, $dmin$
- **Entero:** $numCases$, $i$, $rA$, $sA$, $kA$, $j$, $rB$, $sB$, $kB$, $cw$, $iA$, $iB$, $op$
- $e \leftarrow 10^{-10}$
- **Lea** $numCases$
- **Para**$(i = 1, numCases, 1)$**Haga**
  - // Lectura de la información del avión $A$
  - **Lea** $rA, sA, kA$
  - $sumt \leftarrow 0$
  - **Lea** $x, y, z$
  - $A[1, 1] \leftarrow x, A[1, 2] \leftarrow y, A[1, 3] \leftarrow z, A[1, 4] \leftarrow sumt$
  - **Para**$(j = 2, kA, 1)$**Haga**
    - **Lea** $x, y, z$

1 2 3

1 2 3

$d \leftarrow \sqrt{(x - A[j-1,1])^2 + (y - A[j-1,2])^2 + (z - A[j-1,3])^2}$
$sumt \leftarrow sumt + d/sA$
$A[j,1] \leftarrow x, A[j,2] \leftarrow y, A[j,3] \leftarrow z, A[j,4] \leftarrow sumt$
**Fin_Para**
$d \leftarrow \sqrt{A[kA,1]^2 + A[kA,2]^2 + A[kA,3]^2}$
$sumt \leftarrow sumt + d/sA$
$A[kA+1,1] \leftarrow 0, A[kA+1,2] \leftarrow 0, A[kA+1,3] \leftarrow 0$
$A[kA+1,4] \leftarrow sumt$

// Lectura de la información del avión $B$
**Lea** $rB, sB, kB$
$sumt \leftarrow 0$
**Lea** $x, y, z$
$B[1,1] \leftarrow x, B[1,2] \leftarrow y, B[1,3] \leftarrow z, B[1,4] \leftarrow sumt$
**Para**$(j = 2, kB, 1)$**Haga**
  **Lea** $x, y, z$
  $d \leftarrow \sqrt{(x - B[j-1,1])^2 + (y - B[j-1,2])^2 + (z - B[j-1,3])^2}$
  $sumt \leftarrow sumt + d/sB$
  $B[j,1] \leftarrow x, B[j,2] \leftarrow y, B[j,3] \leftarrow z, B[j,4] \leftarrow sumt$
**Fin_Para**
$d \leftarrow \sqrt{B[kB,1]^2 + B[kB,2]^2 + B[kB,3]^2}$
$sumt \leftarrow sumt + d/sA$
$B[kB+1,1] \leftarrow 0, B[kB+1,2] \leftarrow 0, B[kB+1,3] \leftarrow 0$
$B[kB+1,4] \leftarrow sumt$

$cw \leftarrow 0$
$iA \leftarrow 0, iB \leftarrow 0$
$t1 \leftarrow 0.0, t2 \leftarrow 0.0$
**Mq**$(iA \neq kA$ y $iB \neq kB)$**Haga**
  $t1 \leftarrow t2$
  **Si**$(A[iA+1,4] < B[iB+1,4])$**Entonces**
    $t2 \leftarrow A[iA+1,4]$
    $op \leftarrow 1$
  **Sino Si**$(A[iA+1,4] > B[iB+1,4])$
    $t2 \leftarrow B[iB+1,4]$
    $op \leftarrow 2$
  **Sino**
    $t2 \leftarrow B[iB+1,4]$
    $op \leftarrow 3$
  **F.Si**

  $DsxA \leftarrow (A[iA+1,1] - A[iA,1])/(A[iA+1,4] - A[iA,4])$
  $DsxB \leftarrow (B[iB+1,1] - B[iB,1])/(B[iB+1,4] - B[iB,4])$

1 2 3

1 2 3

$Cx \leftarrow (A[iA,1] - B[iB,1]) - (DsxA * A[iA,4] - DsxB * B[iB,4])$
$Mx \leftarrow DsxA - DsxB$

$DsyA \leftarrow (A[iA+1,2] - A[iA,2])/(A[iA+1,4] - A[iA,4])$
$DsyB \leftarrow (B[iB+1,2] - B[iB,2])/(B[iB+1,4] - B[iB,4])$
$Cy \leftarrow (A[iA,2] - B[iB,2]) - (DsyA * A[iA,4] - DsyB * B[iB,4])$
$My \leftarrow DsyA - DsyB$

$DszA \leftarrow (A[iA+1,3] - A[iA,3])/(A[iA+1,4] - A[iA,4])$
$DszB \leftarrow (B[iB+1,3] - B[iB,3])/(B[iB+1,4] - B[iB,4])$
$Cz \leftarrow (A[iA,3] - B[iB,3]) - (DszA * A[iA,4] - DszB * B[iB,4])$
$Mz \leftarrow DszA - DszB$
**Si**$(Mx^2 + My^2 + Mz^2 > 0)$**Entonces**
$tmin \leftarrow -(Cx * Mx + Cy * My + Cz * Mz)/(Mx^2 + My^2 + Mz^2)$
$dmin1 \leftarrow \sqrt{(Mx * t1 + Cx)^2 + (My * t1 + Cy)^2 + (Mz * t1 + Cz)^2}$
$dmin2 \leftarrow \sqrt{(Mx * t2 + Cx)^2 + (My * t2 + Cy)^2 + (Mz * t2 + Cz)^2}$
$dmin \leftarrow \sqrt{(Mx * tmin + Cx)^2 + (My * tmin + Cy)^2 + (Mz * tmin + Cz)^2}$
**Si**$(tmin > t2)$**Entonces**
$dmin \leftarrow dmin2$
**Sino Si**$(tmin < t1)$
$dmin \leftarrow dmin1$
**F.Si**
**Sino**
$dmin \leftarrow \sqrt{Cx^2 + Cy^2 + Cz^2}$
$dmin1 \leftarrow dmin$
**F.Si**

**Si**$(dmin \leq rA + rB + e \text{ y } (dmin1 > rA + rB + e \text{ o } t1 = 0))$**Entonces**
$cw \leftarrow cw + 1$
**F.Si**

**Si**$(op = 1)$**Entonces**
$iA \leftarrow iA + 1$
**Sino Si**$(op = 2)$
$iB \leftarrow iB + 1$
**Sino**
$iA \leftarrow iA + 1$
$iB \leftarrow iB + 1$
**F.Si**
**Fin_Mq**
**Escriba** $cw$
**Fin_Para**
**Fin_proc**

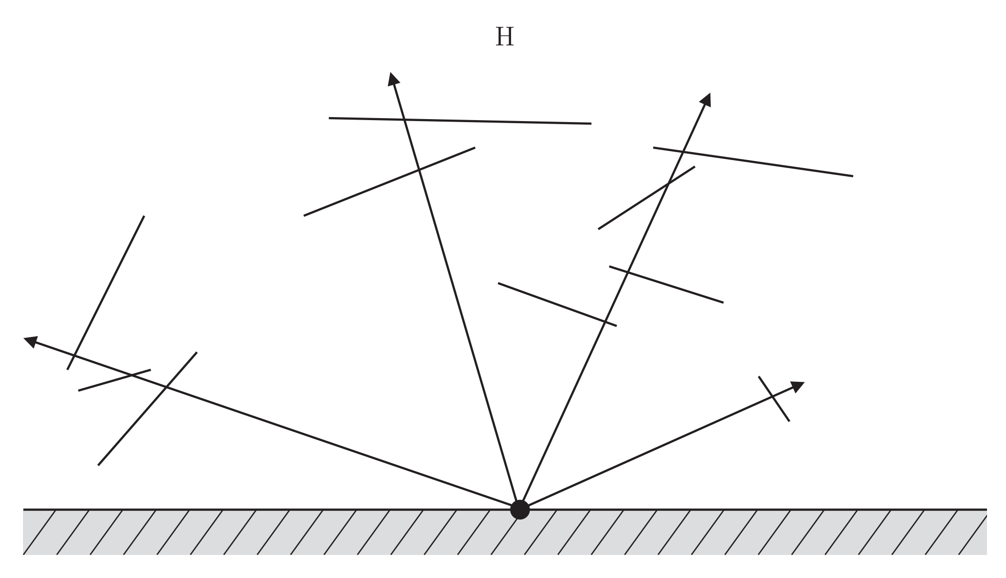



*Figura 6.5* Deporte de tiro

## 6.9 Deporte de tiro

### Enunciado

Fuente: UVA Online Judge, Problema 12323

Cristales Uninorte ofrece a los francotiradores la oportunidad de tiro deportivo en los barrios periféricos de Barranquilla. Un popular tiro deportivo para un francotirador es destruir todos los carteles de cristal en frente de un edificio. La regla que se debe cumplir consiste en que el francotirador debe fijar su pistola en el suelo de la cubierta del edificio, de modo que puede girar libremente en cualquier dirección. Sin embargo, una vez que la pistola se deja en cierto ángulo, no puede ser colocada en cualquier otra ubicación. El objetivo del juego es destruir todas las vallas de cristal haciendo el menor número de disparos.

Cada vez que el concursante dispara, cualquier cristal que la bala toque se destruye, inclusive si la bala toca solo un extremo. Una bala nunca cambia de dirección o velocidad una vez que se dispara, incluso si se destruye cualquier cantidad de vallas de cristal.

La figura 6.5 ilustra un francotirador, en una ubicación fija, frente a un diseño de diez carteles de cristal. El francotirador ha destruido los diez carteles de cristal y ha logrado el objetivo del juego porque los destruyó todos haciendo el menor número de disparos posibles.

Dada la ubicación inicial de un francotirador y un diseño de paneles de cristal,

su tarea consiste en determinar el mínimo número de disparos que destruiría todas las vallas de cristal.

**Entrada**

La primera línea de entrada contiene un número natural $B$, que define el número de paneles de cristal que se encontrarán para ser alcanzados por el francotirador $(1 \leq B \leq 10^3)$. Cada una de las siguientes $B$ líneas contiene cuatro números enteros, $x_1$, $y_1$, $x_2$, $y_2$, separados por espacios en blanco, que definen las coordenadas del segmento de línea con extremos $(x_1, y_1)$ y $(x_2, y_2)$ $(-10^3 \leq x_1, x_2 \leq 10^3, 0 < y_1, y_2 \leq 10^3, y_1 \cdot x_2 \neq x_1 \cdot x_2 y_2)$. Usted puede asumir que el campo de tiro está modelado como la región encima del eje $x$ del plano cartesiano y que el francotirador se encuentra ubicado en el origen $(0, 0)$. A su vez, también puede asumir que los extremos de los segmentos que representan a los paneles de cristal no son colineales con el origen y que ningún par de cristales se intersectan. El último caso está seguido por una línea que contiene dos ceros.

**Salida**

Para cada caso de prueba debe imprimir una línea con el menor número de disparos necesarios del francotirador que destruirá todos los paneles de cristal en el campo de tiro.

**Entrada Ejemplo**

```
10
-309 98 -258 204
-303 83 -251 98
-218 111 -287 31
-145 204 -23 257
-129 272 59 272
-8 159 74 130
150 146 68 174
59 196 128 242
98 256 241 235
197 61 173 92
3
-100 10 -100 50
-50 100 50 100
100 10 100 50
5
-100 100 100 100
-80 120 80 120
```

-60 140 60 140
-40 160 40 160
-20 180 20 180
2
-50 50 0 50
-10 70 50 70
2
-50 50 0 50
0 70 50 70
2
-50 50 0 50
10 70 50 70
0

**Salida Ejemplo**

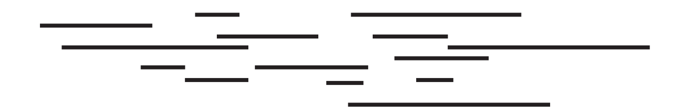
4
3
1
1
1
2

## Análisis

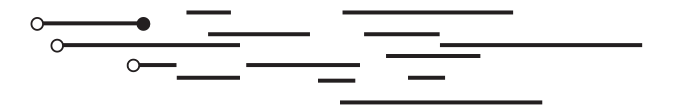
Para garantizar que sea el mínimo número de disparos se debe seguir una estrategia similar a la del problema de Estaciones de Gas. Primero, a los extremos de cada segmento se les asignará un ángulo $crtht[i]$, un variable booleana $crsw[i]$ (que dirá si ese extremo es donde inicia o termina el ángulo) y un $crid[i]$, que corresponde con el segmento. Los extremos deben ser ordenados de mayor a menor respecto al siguiente criterio: un extremo $i$ es mayor que otro $j$ si $crtht[i] > crtht[j]$ o $crsw[i] = falso$ en caso de que $crtht[i] = crtht[j]$.

Una vez realizado el ordenamiento, se toma el primer paréntesis tal que $crsw[i] = verdadero$ y no haya sido eliminado, y se borran todos los que estén debajo de él.

Para ilustrar el procedimiento anterior suponga que cada uno de los segmentos se ha transformado en lo siguiente:



Entonces, el algoritmo buscará la esquina derecha más cercana a la izquierda; en este caso es la señalada con un círculo negro; en el algoritmo eso corresponde a *verdadero*.



Y borrará todos los segmentos que tengan esquinas izquierdas o derechas a su izquierda, es decir, se borrarán los segmentos señalados con un círculo blanco. Una vez eliminados, se repite el procedimiento anterior y se obtiene

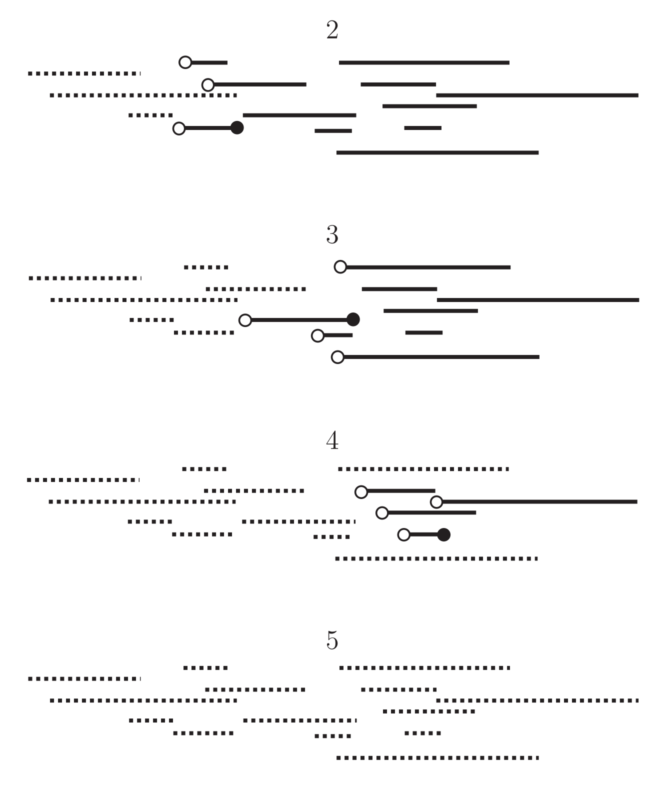



*Figura 6.6* Secuencia del algoritmo

Por último, cada vez que se encuentra un nuevo extremo 'derecho' se incrementa el contador *sum*, que almacena el número mínimo de disparos necesarios para eliminar todos los paneles de cristal.

## Solución

```
Proc: Deporte de tiro
| Entero: numCrystal, i, x1, y1, x2, y2, crid[50], sum
| Real: tht1, tht2, aux, crtht[50],
| Booleano: crsw[50], vivoCr[50]
| numCrystal ← 1
| Mq(numCrystal ≠ 0)Haga
| | Lea numCrystal
| | Si(numCrystal ≠ 0)Entonces
| | | Para(i = 1, numCrystal, 1)Haga
| | | | Lea x1, y1, x2, y2
| | | | tht1 ← atan2(x1, y1)
| | | | tht2 ← atan2(x2, y2)
| | | | Si(tht1 > tht2)Entonces
| | | | | aux ← tht1
| | | | | tht1 ← tht2
| | | | | tht2 ← aux
| | | | F.Si
| | | | crtht[2 * i − 1] ← tht1
| | | | crtht[2 * i] ← tht2
| | | | crsw[2 * i − 1] ← verdadero
| | | | crsw[2 * i] ← falso
| | | | crid[2 * i − 1] ← i − 1
| | | | crid[2 * i] ← i − 1
| | | | vivoCr[i] ← verdadero
| | | Fin_Para
| | | OrdenarCr(crtht, crsw, crid, 2 * numCrystal)
| | | sum ← 0
| | | Para(j = 1, 2 * numCrystal, 1)Haga
| | | | Si(crsw[j] = verdadero y vivoCr[crid[j]] = verdadero)Entonces
| | | | | Para(k = 1, j, 1)Haga
| | | | | | vivoCr[crid[k]] ← falso
| | | | | Fin_Para
| | | | | sum ← sum + 1
| | | | F.Si
| | | Fin_Para
| | | Escriba sum
| | F.Si
| Fin_Mq
Fin_proc

atan2(x, y)
| Real: PI, X, Y, theta
| PI ← 3.1415926535897932
1
```

```
1
| Si(x = 0)Entonces
| | Si(y > 0)Entonces
| | | Devuelva PI/2.0
| | F.Si
| | Si(y < 0)Entonces
| | | Devuelva 3.0 * PI/2.0
| | F.Si
| Sino
| | X ← x, Y ← y
| | theta ← atan(Y/X)
| | Si(x < 0)Entonces
| | | Devuelva theta + PI
| | Sino Si(y < 0)
| | | Devuelva theta + 2.0 * PI
| | Sino
| | | Devuelva theta
| | F.Si
| F.Si
Fin_atan2
```

```
OrdenarCr(crtht, crsw, crid, nc)
| Entero: i, j, aux3
| Booleano: sw, aux2
| Real: aux1,
| Para(i = 1, nc − 1, 1)Haga
| | Para(j = i + 1, nc, 1)Haga
| | | sw ← falso
| | | Si(crtht(i) > crtht(j))Entonces
| | | | sw ← verdadero
| | | Sino Si(crtht(i) = crtht(j) y crsw(i) = falso)
| | | | sw ← verdadero
| | | F.Si
| | | Si(sw = falso)Entonces
| | | | aux1 ← crtht[i]
| | | | crtht[i] ← crtht[j]
| | | | crtht[j] ← aux1
| | | |
| | | | aux2 ← crsw[i]
| | | | crsw[i] ← crsw[j]
| | | | crsw[j] ← aux2
| | | |
| | | | aux3 ← crid[i]
1 2 3 4
```

```
1 2 3 4
│ │ │ │ crid[i] ← crid[j]
│ │ │ │ crid[j] ← aux3
│ │ │ F.Si
│ │ Fin_Para
│ Fin_Para
Fin_OrdenarCr
```

# 6.10 Corte de pizza

## Enunciado

Fuente: UVA Online Judge, Problema 10079

Cuando alguien llama a Ivan "perezoso", dice que es su inteligencia la que le hace ser así. Si su inteligencia le impulsa a hacer algo con el menor esfuerzo físico, ¿por qué debería esforzarse más? También afirma que siempre usa su cerebro y trata de hacer un poco de trabajo con menos esfuerzo; eso no es pereza, sino, más bien, astucia intelectual.

Una vez se le pidió a Ivan que cortara una pizza en siete porciones para distribuirlas entre sus amigos (el tamaño de cada porción de pizza no debía ser igual. De hecho, su porción fue más grande que la de los otros). Pensó un poco y llegó a la conclusión de que se puede cortar en siete porciones usando solo tres cortes rectos con un cuchillo. Siendo así, cortó la pizza de la siguiente manera (adivina cuál es la porción de Ivan):

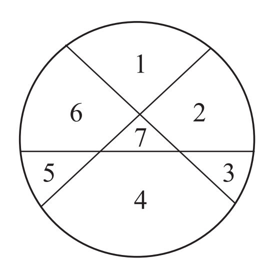



*Figura 6.7* Porciones de pizza

Uno de sus amigos, que nunca creyó en la astucia de Iván, se sorprendió ante dicha sutileza. Pensó: "Si Ivan puede hacerlo, ¿por qué no puede mi ordenador?" Así que trató de hacer un procedimiento similar (pero no exactamente como Iván, por que este le criticaría por haber robado su idea) con su equipo. Escribió un programa que recibe el número de cortes rectos que se desean hacer en la pizza, y

la salida es un número que representa el número máximo de porciones de pizza que se pueden obtener.

Su trabajo consiste en escribir un programa similar. Se le asegura que el amigo de Iván no le molestará por hacer el mismo trabajo que él hizo.

**Entrada**

La entrada consiste en un número entero $N$ $(0 \leq N \leq 210000000)$ por línea. Este número entero representa el número de cortes rectos que se harán en la pizza. Un número negativo determina el fin de la entrada.

**Salida**

La salida por cada entrada debe ser el número máximo de porciones de pizzas que se pueden obtener para el número de cortes ingresado. Cada línea de salida debe contener solo un número entero sin espacios adicionales.

**Entrada Ejemplo**

5
10
-100

**Salida Ejemplo**

16
56

## Análisis

Si se tienen $n$ líneas rectas, el plano se puede dividir con ellas en a lo mucho $\frac{n(n+1)}{2}+1$ regiones diferentes. Note que ninguna recta divide al plano en 1 región, 2 rectas dividen el plano en a lo mucho 2 regiones, 2 rectas en 4 regiones y 3 rectas en 7 regiones. Suponga que $R(n)$ es el número de regiones formadas al dividir el plano con $n$ rectas. Si se traza una nueva recta, no paralela a ninguna de las anteriores trazadas y que no pase por ninguno de los puntos de intersección de las anteriores, entonces habrá $n$ nuevas intersecciones, las cuales formarán $n+1$ nuevas regiones. De lo anterior, $R(n+1) = R(n) + n + 1$. Por lo tanto, para $n$ rectas se tendrá $1 + 2 + \cdots + n + 1 = \frac{n(n+1)}{2} + 1$ regiones.

## Solución

**Proc**: Cortes de pizza
  **Entero:** $n$, $max$
  **Lea** $n$
  **Mq**($n \geq 0$)**Haga**
    $max \leftarrow n(n+1)/2+1$
    **Escriba** $max$
    **Lea** $n$
  **Fin_Mq**
**Fin_proc**

# 6.11 Mancilandia

## Enunciado

Fuente: UVA Online Judge, Problema 11842

Érase una vez Mancilandia, un mundo cuyos habitantes creían que era una región rectangular 2D. Los mancillas (los habitantes de Mancilandia) asumieron que si alguien viaja lo suficiente en una dirección, se caería por el borde de Mancilandia.

Películas Animadas de la Costa (ACM) planea producir una película sobre Mancilandia. Antes de aprobar el guión de la película, ACM quiere familiarizarse con la realidad de Mancilandia mediante la simulación de cómo el mundo pudo evolucionar a partir de las situaciones iniciales dadas y algunas condiciones que determinan la vida y la muerte.

Los mancillas eran nómadas por naturaleza, siempre viajaban a la misma velocidad en líneas rectas, pero cada individuo tenía su propia dirección. Como usted se puede imaginar, si uno observa la vida de un mancilla solo él/ella llegaba a alcanzar un borde de Mancilandia y, por ende, moría. No obstante, si dos mancillas colisionaban, entonces su suerte mejoraba, puesto que solo sus direcciones cambiaban: las direcciones resultantes eran las anteriores reflejadas en un espejo que bisectaba el ángulo formado por las antiguas direcciones de los habitantes que se estrellaban. Por lo tanto, un mancilla sobrevivía porque una colisión con otro cambiaba su dirección.

Sin embargo, había malas noticias cuando más de dos mancillas chocaban en un solo punto: en ese caso todos morían justo en ese punto de colisión. Por lo tanto, note que algunos mancillas podían morir al mismo tiempo. Si este era el caso, el último nombre de la lista de los muertos en un punto común de colisión era el recordado en ese momento (para simplificar, se supondrá que utilizan el alfabeto común y el orden lexicográfico). Los sobrevivientes veneraban el nombre de la última persona muerta hasta que aparecía un nuevo último lugar (y algunos mancillas desaparecían).

La película de ACM comienza con una determinada población en Mancilandia, donde se conocen los nombres, posiciones iniciales y direcciones de cada individuo. ACM quiere ayudar a determinar cuál sería el nombre de la última persona muerta en la vida de toda Mancilandia.

**Entrada**

En total se tienen $NC$ casos de prueba para resolver, $0 < NC < 100$. La primera línea de entrada contiene $NC$. Luego de ello, para cada caso de prueba se tiene un conjunto de líneas:

- La primera línea contiene un número $n$, el número de habitantes en Mancilandia en el mundo inicial ($1 \leq n \leq 100$).
- La segunda línea contiene dos enteros positivos, $B$ y $H$, separados por un espacio, que representan las dimensiones de Mancilandia ($2 \leq B, H \leq 100$). Las coordenadas de Mancilandia son los puntos $(i, j)$, con $0 \leq i \leq B$, $0 \leq j \leq H$. Los límites de Mancilandia son aquellos puntos con coordenadas de la forma $(0, j), (i, 0), (B, j)$ o $(i, H)$.
- $n$ líneas (una por cada mancilla), con cuatro números y una cadena de caracteres: $x, y, d_1, d_2$ y *name* separados por espacios en blanco; $(x, y)$ representa la posición de un mancilla ($0 < x < B, 0 < y < H$ y dos mancillas no pueden iniciar en la misma posición); $(d_1, d_2)$ representa la dirección, $(d_1, d_2)$ es el punto en algún borde de Mancilandia tal que el mancilla correspondiente se mueve hacía él; *name* es una cadena de caracteres de 1 a 10 caracteres alfabéticos en mayúscula, el cual representa el nombre del mancilla. Usted puede asumir que cada mancilla tiene un nombre único.

**Salida**

Para cada caso de prueba dado se debe imprimir una línea con el nombre de la última persona muerta.

**Entrada Ejemplo**

```
2
2
20 23
1 1 0 0 BOB
3 3 3 0 ALICE
3
20 23
```

2 2 4 0 ALICE
4 2 2 0 BOB
1 3 0 3 CHARLES

**Salida Ejemplo**

ALICE
BOB

## Análisis

Se utilizarán los arreglos $p$, $pname$ y la variable $np$ para almacenar información sobre las personas. El arreglo $p$ es de tamaño $4 \times np$, donde para una persona se almacena su posición actual y la dirección hacia donde se está moviendo. La dirección se almacena siempre como un vector unitario $\hat{\mathbf{n}} = p[i,3]\hat{\mathbf{i}} + p[i,4]\hat{\mathbf{j}}$. A su vez, se almacena en $pname$ el nombre de la persona correspondiente.

Por otra parte, en los arreglos $muerte$, $mname$ y $nm$ se almacena la información acerca de las muertes que ocurrieron hasta el momento. Para una persona $i$, en $muerte[i]$ se almacena el tiempo en que murió, $mname[i]$ almacena el nombre correspondiente y $nm$ es el número de muertes que se han presentado.

Para las colisiones que ocurran, en $tcol[i]$ se almacena el tiempo en el que tuvo lugar la colisión. El lugar de la colisión es $(x, y)$ con $x = puntocol[i,1]$, $y = puntocol[i,2]$. Adicionalmente, $pcol$ es una matriz donde en cada fila de la misma se almacena las dos personas que colisionan en dicho lugar. Si tres personas colisionan en un mismo punto, entonces existirá un $i \neq j$ tal que $tcol[i] = tcol[j]$ y las personas en $pcol[i,1], pcol[i,2], pcol[j,1], pcol[j,2]$ habrán colisionado. Similarmente ocurre si más de tres personas colisionan en un mismo punto; $nc$ es el número de colisiones que han tenido lugar.

El procedimiento base que se va a seguir es determinar cuál es la primera colisión que tendrá lugar; para ello se determinan todas las colisiones, y a cada una de ellas se le asigna un tiempo. Este tiempo es de acuerdo con la distancia que tienen que recorrer aquellos que van a colisionar. A continuación se determinará el tiempo de colisión entre dos personas.

Para cada persona se definirá un vector normal asociado que indicará su dirección en dicho instante; para ello

$$\hat{\mathbf{n}} = \left[ \frac{d_1 - x}{\sqrt{(d_1 - x)^2 + (d_2 - y)^2}}, \frac{d_2 - y}{\sqrt{(d_1 - x)^2 + (d_2 - y)^2}} \right] = [n_x, n_y]$$

Sean las trayectorias de dos personas, $a$ y $b$, las siguientes:

$$
\begin{aligned}
(x,y)_a &= (x_0 + (x_1 - x_0)t,\; y_0 + (y_1 - y_0)t)\\
(x,y)_b &= (x_2 + (x_3 - x_2)s,\; y_2 + (y_3 - y_2)s)
\end{aligned}
$$

Por lo tanto, $t$ y $s$ deben cumplir en el punto de colisión que

$$
\begin{aligned}
x_0 + (x_1 - x_0)t &= x_2 + (x_3 - x_2)s\\
y_0 + (y_1 - y_0)t &= y_2 + (y_3 - y_2)s
\end{aligned}
$$

De donde se forma el sistema

$$
\begin{bmatrix} x_1 - x_0 & -(x_3 - x_2)\\ y_1 - y_0 & -(y_3 - y_2) \end{bmatrix} \begin{Bmatrix} t\\ s \end{Bmatrix} = \begin{Bmatrix} x_2 - x_0\\ y_2 - y_0 \end{Bmatrix}
$$

Del sistema anterior se obtiene $t$ y $s$:

$$
t_0 = \frac{(y_3 - y_2)(x_0 - x_2) - (x_3 - x_2)(y_0 - y_2)}{D}
$$

$$
t_1 = \frac{(x_1 - x_0)(y_2 - y_0) - (y_1 - y_0)(x_2 - x_0)}{D}
$$

donde

$$
D = (y_1 - y_0)(x_3 - x_2) - (y_3 - y_2)(x_1 - x_0)
$$

Si $D = 0$, entonces el sistema tiene una matriz singular, lo cual implica que las líneas son paralelas o son la misma. Observe que si $t_0 = 0$ o $t_1 = 0$, entonces las líneas son las mismas, pero es posible que las dos personas no colisionen, puesto que viajan en sentidos opuestos, alejándose o en el mismo sentido. Para verificar si viajan en el mismo sentido, entonces se verifica si $\hat{\mathbf{n}}_a \cdot \hat{\mathbf{n}}_b = 1$. En caso de no viajar en el mismo sentido, puede que se estén alejando. Sin embargo, si se estuviesen acercando, solo tendrían que recorrer mitad de la distancia que les separa, puesto que tienen la misma velocidad. Por lo tanto, se verifica si sus posiciones son la misma una vez han recorrido la distancia antes mencionada. Esa distancia es

$$
l = \frac{1}{2}\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}
$$

Sus posiciones luego de recorridas esas distancias serían

$$
A = (x_0 + l \cdot n_x,\; y_0 + d \cdot n_y),\; B = (x_0 + l \cdot n_x,\; y_0 + l \cdot n_y)
$$

Una vez hecho lo anterior, se determina la primera colisión que tiene lugar, y a su vez se actualizan las posiciones de cada uno de los habitantes. Así hasta que todos hayan muerto. Aquellos que mueren porque se salen de los límites son controlados por la función *tcaida*.

## Solución

```
Proc: Mancilandia
  Entero: Casos, i, n, W, H, j, p[100, 4], t, nm
  Real: dist,
  Cadena: pname[100],
  Lea Casos
  Para(i = 1, Casos, 1)Haga
    Lea n
    Lea W, H
    Para(j = 1, n, 1)Haga
      Lea p[j, 1], p[j, 2], p[j, 3], p[j, 4]
      // Se normaliza el vector de la dirección
      p[j, 3] ← p[j, 3] − p[j, 1]
      p[j, 4] ← p[j, 4] − p[j, 2]
      dist ← √(p[j, 3]^2 + p[j, 4]^2)
      Lea pname[j]
    Fin_Para
    t ← 0
    nm ← 0    // Se inicia el número de muertes en 0
    Mq(np > 0)Haga
      MatarRebote(p, pname, muerte, mname, np, nm, t, tcol, puntocol, pcol, nc, W, H)
    Fin_Mq
    OrdenarMuertes(muerte, mname, np)
    Escriba mname[nm]
  Fin_Para
Fin_proc

MatarRebote(p, pname, muerte, mname, np, nm, t, tcol, puntocol, pcol, nc, W, H)
  sw ← 0
  Para(i = 1, np, 1)Haga
    Si(colisiona(p, np, i, W, H) = falso)Entonces
      nm ← nm + 1
      muerte[nm] ← t + tcaida(p[i, 1], p[i, 2], p[i, 3], p[i, 4], W, H)
      mname[nm] ← pname[i]
      np ← 0
      Para(j = 1, np, 1)Haga
        Si(i ≠ j)Entonces
          np ← np + 1
          Para(k = 1, 4, 1)Haga
            pnew[np, k] ← p[j, k]
          Fin_Para
        F.Si
      Fin_Para
1 2 3
```

```
1 2 3
| | | Para(j = 1, np, 1)Haga
| | | | Para(k = 1, 4, 1)Haga
| | | | | p[j, k] ← pnew[j, k]
| | | | Fin_Para
| | | Fin_Para
| | | sw ← 1
| | F.Si
| Fin_Para
|
| // De aquí en adelante todos colisionan con alguna de las otras personas
| // Por lo tanto, se inicia el proceso de cada una de las colisiones
| // y se les asigna un tiempo, para que se tenga un orden cronológico
| Para(i = 1, np − 1, 1)Haga
| | Para(k = 1, 4, 1)Haga
| | | pi[k] ← p[i, k]
| | Fin_Para
| | Para(j = i + 1, np, 1)Haga
| | | Para(k = 1, 4, 1)Haga
| | | | pj[k] ← p[j, k]
| | | Fin_Para
| | | aux ← colisionan_dos(pi, i, pj, j, auxtcol, auxpc, W, H)
| | | // Se verifica si el tiempo en el que ocurrirá la colisión es positivo
| | | Si(aux ≠ −1)Entonces
| | | | // Se añade una nueva colisión
| | | | nc ← nc + 1
| | | | tcol[nc] ← auxtcol
| | | | puntocol[nc, 1] ← auxpc[1]
| | | | puntocol[nc, 2] ← auxpc[2]
| | | | pcol[nc, 1] ← i
| | | | pcol[nc, 2] ← j
| | | F.Si
| | Fin_Para
| Fin_Para
|
| OrdenarCol(tcol, puntocol, pcol, nc)
| // Se inicia pmc (personas muertas en colisiones) en 0
| pmc ← 0
| i ← 1
| Mq(i < n y tcol[1] = tcol[i])Haga
| | j ← i
| | Mq(j < n y tcol[i] = tcol[j] y puntocol[i] = puntocol[j])Haga
| | | j ← j + 1
| | Fin_Mq
1 2
```

```
1 2
    Si(j − i = 1)Entonces
        // Entonces solo se trata de dos personas
        // por lo tanto, nadie muere y solo se intercambian
        // sus nombres
        aux ← pname[i]
        pname[i] ← pname[j]
        pname[j] ← aux
    Sino
        // Choque entre mas de dos personas
        Para(k = i, j − 1, 1)Haga
            muerepersona(pmc_vec,pmc,pcol,k)
        Fin_Para
    F.Si
    i ← j
  Fin_Mq

  newp ← 0
  Para(i = 1, np, 1)Haga
    Si(estamuerta(pmc_vec, pmc, i))Entonces
        nm ← nm + 1
        muerte[nm] ← t + tcol[1]
        mname[nm] ← pname[i]
    Sino
        newp ← newp + 1
        newp_vec[newp, 1] ← p[i, 1]
        newp_vec[newp, 2] ← p[i, 2]
        newp_vec[newp, 3] ← p[i, 1] + p[i, 3] ∗ tcol[1]
        newp_vec[newp, 4] ← p[i, 2] + p[i, 4] ∗ tcol[1]
        pnamenew[newp] ← pname[i]
    F.Si
  Fin_Para

  Para(i = 1, newp, 1)Haga
    pname[i] ← pnamenew[i]
    Para(k = 1, 4, 1)Haga
        p[i, k] ← newp_vec[i, k]
    Fin_Para
  Fin_Para
  np ← newp
  t ← t + tcol[1]
Fin_MatarRebote
```

```
colisiona(p, np, i, W, H)
  Para(j = 1, np, 1)Haga
    Si(i ≠ j)Entonces
      Para(k = 1, 4, 1)Haga
        pi[k] ← p[i, k]
        pj[k] ← p[j, k]
      Fin_Para
      aux ← colisionan_dos(pi, i, pj, j, auxtcol, auxpc, W, H)
      Si(aux ≠ −1)Entonces
        Devuelva verdadero
      F.Si
    F.Si
  Fin_Para
  Devuelva falso
Fin_colisiona
colisionan_dos(pi, i, pj, j, auxtcol, auxpc, W, H)
  Si(pi[1] = pj[1] y pi[2] = pj[2])Entonces
    Devuelva −1
  F.Si
  x0 ← pi[1], x1 ← pi[1] + pi[3]
  y0 ← pi[2], y1 ← pi[2] + pi[4]
  x2 ← pj[1], x3 ← pj[1] + pj[3]
  y2 ← pj[2], y3 ← pj[2] + pj[4]
  t0 ← (y3 − y2) * (x0 − x2) − (x3 − x2) * (y0 − y2)
  t1 ← (x1 − x0) * (y2 − y0) − (y1 − y0) * (x2 − x0)
  D ← (y1 − y0) * (x3 − x2) − (y3 − y2) * (x1 − x0)

  Si(D = 0)Entonces
    Si(t0 = 0 o t1 = 0)Entonces
      Si(dot(pi, pj) = 1)Entonces
        Devuelva −1
      F.Si
      // Supuesto punto de colisión
      pcx ← (x0 + x2)/2
      pcy ← (y0 + y2)/2
      l ← √((x2 − x0)² + (y2 − y0)²)
      Si(x0 + l * pi[3] ≠ pcx o y0 + l * pi[4] ≠ pcy)Entonces
        Devuelva −1
      Sino Si(x2 + l * pj[3] ≠ pcx o y2 + l * pj[4] ≠ pcy)
        Devuelva −1
      F.Si
      auxtcol ← l
1 2 3
```

1 2 3

$auxpc[1] \leftarrow pcx$
$auxpc[2] \leftarrow pcy$
Devuelva 0
**F.Si**
// Las líneas son simplemente paralelas, no hay intersección
Devuelva $-1$
**F.Si**

// No son paralelas
$t0 \leftarrow t0/det$
$t1 \leftarrow t1/det$
**Si**($t0 = t1$ y $t0 \geq 0$ y $t1 \geq 0$)**Entonces**
$x \leftarrow x0 + t0 * (x1 - x0)$
$y \leftarrow y0 + t1 * (y1 - y0)$
**Si**(fueradelmundo$(x, y, W, H) = verdadero$)**Entonces**
Devuelva $-1$
**F.Si**
$auxtcol \leftarrow t0$
$auxpc[1] \leftarrow x$
$auxpc[2] \leftarrow y$
Devuelva 0
**F.Si**
**Fin_colisionan_dos**
**dot(**$pi, pj$**)**
Devuelva $pi[3] * pj[3] + pi[4] * pj[4]$
**Fin_dot**
**fueradelmundo(**$x, y, W, H$**)**
**Si**($x < 0$ o $x > W$ o $y < 0$ o $y > H$)**Entonces**
Devuelva $verdadero$
**F.Si**
**Fin_fueradelmundo**
**tcaida(**$x0, y0, x1, y1, W, H$**)**
$x1 \leftarrow x1 + x0$
$y1 \leftarrow y1 + y0$

$xsol \leftarrow -x0/(x1 - x0)$
$ysol \leftarrow y0 + (y1 - y0) * xsol$
**Si**($0 \leq ysol$ y $ysol \leq H$ y $xsol \geq 0$)**Entonces**
Devuelva $\sqrt{(ysol - y0)^2 + (xsol - x0)^2}$
**F.Si**

1

1

$xsol \leftarrow (W - x0)/(x1 - x0)$
$ysol \leftarrow y0 + (y1 - y0) * xsol$
**Si**($0 \leq ysol$ y $ysol \leq H$ y $xsol \geq 0$)**Entonces**
  Devuelva $\sqrt{(ysol - y0)^2 + (xsol - x0)^2}$
**F.Si**

$ysol \leftarrow -y0/(y1 - y0)$
$xsol \leftarrow x0 + (x1 - x0) * ysol$
**Si**($0 \leq xsol$ y $xsol \leq W$ y $ysol \geq 0$)**Entonces**
  Devuelva $\sqrt{(ysol - y0)^2 + (xsol - x0)^2}$
**F.Si**

$ysol \leftarrow (H - y0)/(y1 - y0)$
$xsol \leftarrow x0 + (x1 - x0) * ysol$
**Si**($0 \leq xsol$ y $xsol \leq W$ y $ysol \geq 0$)**Entonces**
  Devuelva $\sqrt{(ysol - y0)^2 + (xsol - x0)^2}$
**F.Si**
**Fin_tcaida**

**OrdenarCol(**$tcol, puntocol, pcol, nc$**)**
  **Para**($i = 1, nc - 1, 1$)**Haga**
    **Para**($j = i + 1, nc, 1$)**Haga**
      **Si**($tcol[i] > tcol[j]$)**Entonces**
        $aux \leftarrow tcol[i]$
        $tcol[i] \leftarrow tcol[j]$
        $tcol[j] \leftarrow aux$
        **Para**($k = 1, 2, 1$)**Haga**
          $aux \leftarrow puntocol[i, k]$
          $puntocol[i, k] \leftarrow puntocol[j, k]$
          $puntocol[j, k] \leftarrow aux$
        **Fin_Para**

        **Para**($k = 1, 2, 1$)**Haga**
          $aux \leftarrow pcol[i, k]$
          $pcol[i, k] \leftarrow pcol[j, k]$
          $pcol[j, k] \leftarrow aux$
        **Fin_Para**
      **F.Si**
    **Fin_Para**
  **Fin_Para**
**Fin_OrdenarCol**

```
muerepersona(pmc_vec,pmc,pcol,k)
| Para(i = 1, 2, 1)Haga
| | sw ← 0
| | Para(j = 1, pmc, 1)Haga
| | | Si(pmc_vec[j] = pcol[k, i])Entonces
| | | | sw ← 1
| | | F.Si
| | Fin_Para
| | Si(sw = 0)Entonces
| | | pmc ← pmc + 1
| | | pmc_vec[pmc] ← pcol[k, i]
| | F.Si
| Fin_Para
Fin_nuevaspersonas

estamuerta(pmc_vec, pmc, k)
| sw ← 0
| Para(i = 1, pmc, 1)Haga
| | Si(pmc_vec[i] = k)Entonces
| | | sw ← 1
| | F.Si
| Fin_Para
| Devuelva sw
Fin_estamuerta

ordenarmuertes(muerte, mname, nm)
| Para(i = 1, nm − 1, 1)Haga
| | Para(j = i + 1, nm, 1)Haga
| | | Si(muerte[i] > muerte[j] o mname[i] > mname[j])Entonces
| | | | aux ← muerte[i]
| | | | muerte[i] ← muerte[j]
| | | | muerte[j] ← aux
| | | |
| | | | aux ← mname[i]
| | | | mname[i] ← mname[j]
| | | | mname[j] ← aux
| | | F.Si
| | Fin_Para
| Fin_Para
Fin_ordenarmuertes
```

# 6.12 Sumando números invertidos

## Enunciado

Los comediantes antiguos de Malidinesia prefieren comedias y tragedias. Desafortunadamente, la mayoría de las antiguas obras son tragedias; por lo tanto, el asesor dramático de ACM ha decidido transfigurar algunas tragedias en comedias. Obviamente, este trabajo es muy duro, porque el sentido básico de la obra debe mantenerse intacto, a pesar de que todas las cosas cambian a sus opuestos. Por ejemplo, los números: si algún número aparece en la tragedia, este debe convertirse a su forma invertida antes de ser aceptado en la obra de teatro de comedia.

Un "número invertido" es un número escrito en números arábigos pero con el orden de los dígitos invertidos. El primer dígito se convierte en el último y viceversa. Por ejemplo, si el héroe principal tenía 1245 fresas en la tragedia, entonces tiene .421 en la nueva obra. Tenga en cuenta que todos los ceros iniciales se omiten. Eso significa que si el número termina con un cero, el cero se pierde por la inversión (por ejemplo, 1200 da 21). También tenga en cuenta que la cantidad invertida no tiene ceros a la derecha, es decir, no es múltiplo de 10.

ACM necesita hacer algunas cuentas con números invertidos. Su tarea consiste en sumar dos números invertidos y mostrar en la salida la suma invertida. Por supuesto, el resultado no es único, ya que cualquier número en particular es una forma invertida de varios números (por ejemplo, 21 podría ser 12, 120 o 1200 antes de la inversión). Por lo tanto, debemos asumir que no hay ceros que se perdieron por la marcha atrás (por ejemplo, suponer que el número original era 12).

### Entrada

La entrada consiste en $N$ casos. La primera línea de la entrada contiene solo un entero positivo $N$. A continuación siguen los casos. Cada caso consta de exactamente una línea con dos enteros positivos separados por un espacio en blanco. Estos son los números invertidos que se deben sumar. Los números serán como máximo de 200 caracteres.

### Salida

Para cada caso imprima exactamente una línea que contiene un solo número entero: la suma invertida de dos números invertidos. Omita los ceros a la izquierda en la salida.

**Entrada Ejemplo**

3
24 1
4358 754
305 794

**Salida Ejemplo**

34
1998
1

## Análisis

Puesto que los números que se van a leer pueden tener hasta 200 caracteres de longitud, se tratarán como cadenas. La ventaja de usar cadenas consiste en que implícitamente tienen funciones como *size* y *erase*, que respectivamente dicen el tamaño que tiene la cadena y borrar caracteres de la misma. Por lo tanto, lo primero que se debe hacer es validar que los números leídos no tengan ceros a la derecha; para ello debe usarse un ciclo Mientras que.

Otra ventaja de trabajar con cadenas consiste en que la concatenación de cadenas es muy sencilla. En este caso la suma es la que sirve de operador como concatenación; por ejemplo, '123' + '45' = '12345'. El procedimiento que se va a seguir entonces es simular la suma de los dos números.

La variable *carry* será la encargada de llevar el acarreo; la variable $x$ llevará la suma temporal de dos dígitos. Las cadenas $r$ y $s$ son los dos números que se van a sumar. La cadena $r$ contiene el resultado. Supongamos que se quiere sumar los números $s = \overline{s_1 \cdots s_k}$ y $t = \overline{t_1 \cdots t_l}$. Si se invierten, entonces $s_1$ se suma con $t_1$, $s_2$ se suma con $t_2$, y así sucesivamente. Por lo tanto, se barrerá la cadena de caracteres, desde 1 hasta el máximo entre $k$ y $l$, que son, respectivamente, $s.size()$ y $t.size()$.

Además, como la idea es que la suma también se invierta, esto se hará de la siguiente forma: primero se realiza una suma $s_i$ con $t_i$ y se guarda el acarreo de esto. A lo que vaya en $r$ se le suma esto en forma de cadenas, de tal forma que el resultado siempre se está concatenando a la derecha, no a izquierda. Lo anterior implica que la suma ya está siendo invertida automáticamente.

Finalmente se imprime $r$, que es la suma invertida.

## Solución

**Proc**: Suma de números invertidos
    **Entero:** $S$, $i$, $n$, $carry$, $x$
    **Cadena:** $s$, $t$, $r$
    **Lea** $S$
    **Para**$(i = 1, S, 1)$**Haga**
        **Lea** $s, t$
        $r \leftarrow$ ‘’
        **Mq**$(s.size() > 1$ y $s[s.size()] =$ ‘0’)**Haga**
            $s.erase(s.size())$
        **Fin_Mq**
        **Mq**$(t.size() > 1$ y $t[t.size()] =$ ‘0’)**Haga**
            $t.erase(t.size())$
        **Fin_Mq**
        $n \leftarrow \max(s.size(), t.size())$
        $carry \leftarrow 0$
        **Para**$(i = 1, n, 1)$**Haga**
            $x \leftarrow carry$
            **Si**$(i \leq s.size())$**Entonces**
                $x \leftarrow x + s[i] -$ ‘0’
            **F.Si**
            **Si**$(i \leq t.size())$**Entonces**
                $x \leftarrow x + t[i] -$ ‘0’
            **F.Si**
            $carry \leftarrow x/10$
            $x \leftarrow x$ **mod** $10$
            $r \leftarrow r + x +$ ‘0’
        **Fin_Para**
        **Si**$(carry > 0)$**Entonces**
            $r \leftarrow r + carry +$ ‘0’
        **F.Si**
        **Mq**$(r.size() > 1$ y $r[0] =$ ‘0’)**Haga**
            $r.erase(0, 1)$
        **Fin_Mq**
        **Escriba** $r$
    **Fin_Para**
**Fin_proc**

# 6.13 Jessica y los números impares

A Jessica le encanta jugar con los números impares. El otro día comenzó a escribir en cada línea un número impar de números impares. La lista es similar a esta:

1
3 5 7
9 11 13 15 17
19 21 23 25 27 29 31
...

En cierta línea Jessica escribió 55 números impares. ¿Se puede descubrir la suma de los tres últimos números escritos en esa línea? ¿Se puede hacer esto en general para una cantidad determinada de números impares?

Dado el número $N$ de los números impares en una determinada línea, su tarea consiste en determinar la suma de los tres últimos números de esa línea.

**Entrada**

La entrada es una secuencia de líneas, en la que en cada línea hay un número impar $N$ $(1 < N < 1000000000)$. Al último caso de prueba le sigue un $-1$.

**Salida**

Para cada línea de entrada escriba la suma de los tres últimos números impares escritos por Jessica en esa línea de $N$ números. Esta suma se garantiza que es menor que $2^{63}$.

**Entrada Ejemplo**

3
5
7
-1

**Salida Ejemplo**

15
45
87

## Análisis

Note que 1 es el número impar número 1, 7 es el número impar número $1 + 3$, es decir, el número 4, que 17 es el número impar número $1 + 3 + 5 = 9$. Por lo tanto, luego de escribir 1 fila se ha escrito 1 número impar; luego de escribir 2 filas

se han escrito 4 números impares; 3 implica 9 números impares. En general, luego de escribir $n$ filas se han escrito $n^2$ números impares. Esto se debe a que la suma de los primeros $k$ números impares es $k^2$; por ejemplo:

$$\begin{aligned} 1 &= 1 = 1^2 \\ 1+3 &= 4 = 2^2 \\ 1+3+5 &= 9 = 3^2 \\ 1+3+5+7 &= 16 = 4^2 \\ 1+3+\cdots+(2k-1) &= k^2 \end{aligned}$$

Por lo tanto, si $2k-1=N$ (pues en la última fila se escriben $2k-1$ impares), entonces $k=(N+1)/2$; de donde se tiene que el último número escrito es el impar número $k^2$, es decir, $(N+1)^2/4$. Lo anterior implica que si $i=(N+1)^2/4$, entonces la suma buscada es

$$(2i-5)+(2i-3)+(2i-1)=6i-9.$$

## Solución

```
Proc: Números impares
| Entero: n, i
| n ← 1
| Mq(n > 0)Haga
| | Lea n
| | Si(n > 0)Entonces
| | | i ← (n + 1)^2/4
| | | Escriba 6 * i − 9
| | F.Si
| Fin_Mq
Fin_proc
```

# 6.14 Mezclando invitaciones

## Enunciado

Fuente: UVA Online Judge, Problema 11282

Kelly tendrá pronto una fiesta de cumpleaños, y quisiera invitar a todos sus amigos. Sin embargo, hay un problema: ¡su casa es demasiado pequeña para dar cabida a todo el mundo! Ya ha escrito las invitaciones personalizadas a todos sus amigos, con sus correspondientes sobres, para enviarlos por correo, y ahora no sabe qué hacer.

Por suerte, Alex ha pensado en una forma inteligente de ayudar a Kelly en la tarea. Él sabe que si un amigo recibe una invitación personalizada de él o ella, ese amigo sin duda vendrá a la fiesta. Sin embargo, si un amigo recibe una invitación personalizada de otra persona (es decir, la invitación equivocada), ese amigo seguramente no vendrá a la fiesta. Alex va a mezclar invitaciones y sobres de Kelly para que algunas invitaciones sean enviadas por correo al destinatario equivocado. De esta manera, todo el mundo todavía tiene una invitación y nadie se quede fuera, pero no irá más gente de la que cabe en la casa de Kelly.

Hay exactamente tantos sobres como invitaciones, y cada invitación tiene un sobre correspondiente con el mismo destinatario. Alex debe colocar cada invitación en un sobre de tal manera que no haya más invitaciones en los sobres correctos que la cantidad de personas que Kelly puede acomodar en su casa.

Siendo el matemático inteligente que es, Alex le gustaría contar el número de formas en que se puede mezclar las invitaciones y los sobres para lograr su objetivo. ¿Se puede escribir un programa para hacer esto para Alex?

**Entrada**

El archivo de entrada contiene varios casos de prueba, cada uno en una línea separada. Cada línea contiene dos números enteros positivos, $N$ y $M$, separadas por un espacio. $N$ $(1 \leq N \leq 20)$ es el número de invitaciones que ha escrito Kelly, y $M$ $(0 \leq M \leq N)$ es el número máximo de personas que puede invitar. El último caso de prueba está seguido por dos 0 separados por un espacio en blanco.

**Salida**

Para cada caso de prueba escriba en una línea independiente un entero: el número de maneras en que Alex puede mezclar las invitaciones y los sobres para lograr su objetivo.

**Entrada Ejemplo**

3 2
4 1
4 4
0 0

**Salida Ejemplo**

5
17
24

## Análisis

Sea $f(n,k)$ el número de permutaciones de $n$ elementos tales que $k$ de ellos están en la posición correcta. Por ejemplo, $f(3,1)=3$, y las permutaciones son $(1,3,2)$, $(3,2,1)$ y $(2,1,3)$. Siendo así, la respuesta al problema sería

$$\sum_{k=0}^{m} f(n,k)$$

Esto se debe a que Kelly mínimo puede invitar a nadie o máximo a las $m$ personas que pueden entrar en su casa.

Por otro lado, sea $D(n)$ el número de desarreglos de $n$ elementos, es decir, el número de permutaciones en las que ningún elemento está en su posición original. Por ejemplo, $D(3)=2$, las cuales son $(2,3,1)$ y $(3,1,2)$. Note que para cada $n$, $f(n,0)=D(n)$, es decir, ningún elemento queda en su posición original. En general,

$$f(n,k)=\binom{n}{k}\cdot D(n-k)$$

La idea detrás de esto es que se escogen de las $n$ las $k$ personas que van a quedar en su posición correcta y se desordena el resto. Note que para escoger $k$ personas de $n$ se tiene $\binom{n}{k}$ formas y para desordenar el resto se tiene $D(n-k)$ formas, de allí se sigue el resultado.

Por último, hay diferentes formas de calcular $D(n)$; mediante el Principio de Inclusión-Exclusión se tiene que

$$D(n)=n!\left(\frac{1}{0!}-\frac{1}{1!}+\cdots+(-1)^n\frac{1}{n!}\right).$$

Sin embargo, también es posible determinar $D(n)$ recursivamente mediante

$$D(n)=(n-1)(D(n-1)+D(n-2)).$$

Para comprender este resultado, note que para que el primer elemento no esté en su posición correcta, este debe moverse a las otras $n-1$ posiciones restantes, por lo tanto, esto lo puede hacer de $n-1$ formas. Suponga que este elemento se mueve a la posición $i$, entonces existen dos posibilidades de acuerdo a si el elemento en la posición $i$ se mueve a la posición 1 o no.

- El elemento en la posición $i$ no se mueve a la posición 1. Este caso es equivalente a resolver el problema de desarreglos de $n-1$: cada una de las personas tiene que moverse a una posición que no sea la suya; en el caso del elemento $i$, su posición prohibida es la 1. De lo anterior se tiene $D(n-1)$ formas para este caso.

- El elemento en la posición $i$ se mueve a la posición 1. Note que esto reduce el problema al caso de $n-2$, es decir, hay $D(n-2)$ formas de hacer esto.

Finalmente se consigue el resultado.

## Solución

```
Proc: Mezclando invitaciones
| Entero: N, i, choose[50, 50], j, d[50], n, m, ans
| Lea N
| // Coeficientes binomiales
| Para(i = 0, N, 1)Haga
| | choose[i, 0] ← 1
| | choose[i, i] ← 1
| Fin_Para
| Para(i = 1, N, 1)Haga
| | Para(j = 1, i − 1, 1)Haga
| | | choose[i, j] ← choose[i − 1, j − 1] + choose[i − 1, j]
| | Fin_Para
| Fin_Para
| // Desarreglos
| d[0] ← 1
| d[1] ← 0
| Para(i = 2, N, 1)Haga
| | d[i] ← (i − 1) ∗ (d[i − 2] + d[i − 1])
| Fin_Para
|
| n ← 1
| m ← 1
| Mq(n ≠ 0 o m ≠ 0)Haga
| | Lea n, m
| | Si(n ≠ 0 o m ≠ 0)Entonces
| | | ans ← 0
| | | Para(k = 0, m, 1)Haga
| | | | ans ← ans + choose[n, k] ∗ d[n − k]
| | | Fin_Para
| | | Escriba ans
| | F.Si
| Fin_Mq
Fin_proc
```

# 6.15 Contando combinaciones de bloques

## Enunciado

### Parte 1

Una fila que mide siete unidades de longitud cuenta con bloques de color rojo con una longitud mínima de tres unidades, de forma que cualquiera de los dos bloques de color rojo (que pueden ser de diferentes longitudes) está separado por al menos un cuadrado negro. Hay exactamente diecisiete maneras de hacer esto.

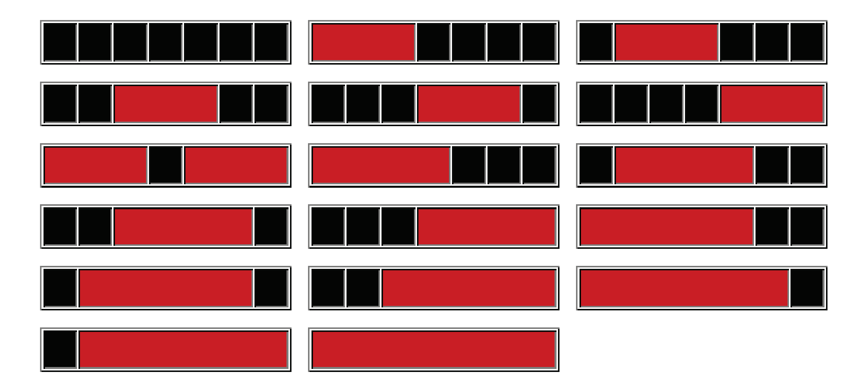


*Figura 6.8* Fila de 7 unidades de longitud

Teniendo en cuenta lo antes mencionado, ¿de cuántas maneras se puede rellenar una fila de cincuenta unidades de longitud?

NOTA: Aunque el ejemplo anterior no se presta para tal posibilidad, en general se permite mezclar tamaños de bloques. Por ejemplo, en una fila de ocho unidades de medida de longitud se puede utilizar uno rojo (3), uno negro (1) y otro rojo (4).

### Parte 2

El problema anterior es posible volverlo más general. Suponga que tiene una fila de $n$ unidades de longitud y bloques de color rojo con una longitud de mínimo $m$ unidades ubicados sobre la fila tal que cualesquiera dos bloques rojos (los cuales pueden tener diferente longitud) están separados por al menos un bloque negro.

Sea la $F(m, n)$ la función que cuenta el número de formas de llenar la fila.

Por ejemplo, $F(3, 29) = 673135$ y $F(3, 30) = 1089155$.

Eso es, para $m = 3$ puede notarse que $n = 30$ es el menor valor para el cual $F(m, n)$ es mayor que un millón.

De la misma forma, si $m = 10$, puede verificarse que $F(10, 56) = 880711$ y $F(10, 57) = 1148904$, de tal manera que $n = 57$ es el menor valor para el cual $F(m, n)$ es mayor que un millón.

Determine para $m = 50$ el menor valor de $n$ para el cual $F(m, n)$ es mayor que un millón.

## Análisis

Se demostrará que $F(m, n) = 2F(m, n-1) - F(m, n-2) + F(m, n-m-1)$. Para ello se numerarán cada uno de los bloques de la fila. Note que para rellenar la fila de $n$ bloques se tiene varias posibilidades:

- Que el primer bloque no se rellena. De esta forma restan $n-1$ bloques por rellenar, y esto se puede hacer de $F(m, n-1)$ formas.
- Si se rellena el primer bloque:
  - Se puede ubicar una ficha de longitud $m$. Sin embargo, el siguiente bloque tiene que ser negro, por lo tanto, resta rellenar una fila de $n-m-1$ bloques, lo cual se puede hacer de $F(m, n-m-1)$ formas.
  - Se puede ubicar una ficha de longitud $m+1$. Similar al caso anterior, esto se puede hacer de $F(m, n-m-2)$ formas.
  - En general, se puede ubicar una ficha de longitud $m+k$, con $k \geq 0$ y $m+k \leq n$, lo cual se puede hacer de $F(m, n-m-k-1)$ formas.

De lo anterior se tiene que

$$F(m, n) = F(m, n-1) + F(m, n-m-1) + F(m, n-m-2) + \cdots + F(m, 0).$$

Si se aplica esta fórmula para $F(m, n-1)$ se tiene

$$F(m, n-1) = F(m, n-2) + F(m, n-m-2) + \cdots + F(m, 0)$$

Restando del primer resultado el segundo se tiene

$$F(m, n) - F(m, n-1) = F(m, n-1) + F(m, n-m-1) - F(m, n-2)$$

de donde

$$F(m, n) = 2F(m, n-1) - F(m, n-2) + F(m, n-m-1).$$

Si se desea que $n-m-1 > 0$, entonces $n > m+1$, por lo tanto, se debe conocer $F(m, n)$ para $n = 1, 2, \ldots, m+1$. Se debe calcular $F(m, n)$ para estos valores. Es fácil ver que $F(m, n) = 1$ para $1 \leq n < m$ y que $F(m, m) = 2$. Note que $F(m, m+1) = 4$, puesto que existen 4 posibilidades de llenar una fila de $m+1$ bloques con bloques con longitud mínima igual a $m$:

- No colocar ningún bloque. Esto se puede hacer solo de una forma.
- Colocar un bloque de longitud $m$. Esto se puede hacer de dos formas: dejando el espacio en blanco al inicio o al final.
- Colocar un bloque de longitud $m + 1$. Esto se puede hacer solo de una forma.

Con lo anterior se puede determinar las respuestas a la parte 1 y 2 del problema. Al final de la solución de la Parte 1 se presenta el valor de $F(3, 50)$.

## Solución

### Parte 1

```
Proc: Contando
  Entero: m, n, i, formas[50]
  // Se declaran m y n
  m ← 3
  n ← 50
  Para(i = 1, m − 1, 1)Haga
    formas[i] ← 1
  Fin_Para
  formas[m] ← 2
  formas[m + 1] ← 4
  formas[m + 2] ← 7
  Para(i = m + 3, n, 1)Haga
    formas[i] ← 2 ∗ formas[i − 1] − formas[i − 2] + formas[i − m − 1]
  Fin_Para
  Escriba formas[n]
Fin_proc
```

Solución Parte 1: 16475640049.

Se deja como ejercicio al lector resolver la Parte 2.

# Resumen

El avance del conocimiento humano ha permitido logros importantes para la sociedad desde la edad antigua hasta la era actual, enmarcada en el paradigma socio-técnico conocido como la Sociedad del Conocimiento en un mundo globalizado. Paradigma que es técnico por cuanto las Tecnologías de la Información y las Comunicaciones (TIC) afectan y afectarán las actuales y futuras estructuras de la sociedad en los contextos políticos, económicos y sociales.

Las TIC soportadas por la macrored Internet están fundamentadas en las máquinas de procesamiento electrónico de datos (el computador) y las redes de ordenadores. La operación de sistemas informáticos soportados por la red tiene su fundamento en el software, el cual funciona con el *logiciel* o la lógica de control contenida en el programa que corre en un computador. La lógica de programación está fundamentada en una lógica algorítmica; y esta, a su vez, solo puede ser generada mediante una lógica humana racional. Luego, el estudio de los fundamentos básicos para la construcción, análisis y diseño de algoritmos representativos de las interrelaciones de la lógica humana-algorítmica a convertirse en lógica de programación con base en lo desarrollado en este texto permite concluir:

- Es necesario el concepto de lógica como núcleo fundamental del estudio de los algoritmos; o lo que es equivalente, son necesarias formas válidas de deducción representadas por reglas inferenciales que perteneciendo al sistema de razonamiento humano permiten pasar las estructuras lógicas construidas en la mente del sujeto a través de sus reglas deductivas a reglas algorítmicas que representan la lógica de la resolución de un problema factible de ser tratado por la algoritmia. La simbiosis de la lógica humana y la lógica algorítmica, generan por lo tanto, la lógica de programación.

- Las simbiosis de las lógicas humana y algorítmica se realiza por humanos y para humanos, de ahí la importancia de la lógica humana-algorítmica racional; este concepto induce la comprensión del impacto de las soluciones contenidas en la lógica de programación operada en la red en contextos globales, económicos, ambientales y sociales.

- En su desarrollo la lógica no ha sido ajena al desarrollo histórico de las generaciones de seres humanos; por lo tanto, el desarrollo histórico de los algoritmos permite entender el avance de la lógica algorítmica con base en la racionalidad

humana; punto en el cual se entraman otros campos del conocimiento humano relacionados con la algorítmia, como son la Matemática y la Ciencia de la Computación (*Computer Science*).

- El fundamento lógico humano-algorítmico, unido a los conceptos matemáticos y de la Ciencia de la Computación, ha permito la construcción de lógicas de programación más robustas, lo cual ha sido posible solo mediante el diseño y análisis de algoritmos representativos a la resolución de problemas de cualquier área del conocimiento humano; tal es el caso de los sistemas predictivos climatológicos o sistemas traductores multilingües entre otros.

- La lógica algorítmica en su generación de lógica de programación es aplicable a cualquier campo del conocimiento humano; en tal sentido, las áreas económicas, ambientales, médicas, jurídicas y económicas se han servido de la algoritmia; pero una de las áreas que mayor influencia ha recibido de la algoritmia es la Ingeniería; justificado por el hecho de que las acciones teóricas, de abstracción y diseño en las ingenierías dependen necesariamente de la matemática, la física y la química, entre otras; áreas que soportadas por la lógica humana-algorítmica y de programación funcionando a través de ordenadores permiten aplicar el conocimiento ingenieril en un proceso de diseño estructurado, a partir de un fundamento teórico complementado con una acción de abstración formal, para construir diseños que permiten la solución de problemas con la participación de la Ingeniería. Luego, la algorítmia es base estructural para generar la lógica de programación; y la integración de las lógicas humana-algorítmica-programable es, a su vez, base en actividades: teóricas, por el acervo de información contenido en la red; de abstracción, porque en sí la construcción de un algoritmo para generar una lógica de programación es un ejercicio de abstracción; y de diseño, por el hecho de que las herramientas fabricadas en software con base en algoritmos son necesarias para apoyar los procesos de diseño en cualquier Ingeniería.

- Reconociendo el impacto que ha tenido y que tendrá la lógica de programación en la Ingeniería, es, por lo tanto, necesario la enseñanza y el aprendizaje de la algoritmia para el desarrollo de la Ingeniería Moderna. El cuerpo teórico de la algoritmia está conformado por estructuras lógicas algorítmicas que se interrelacionan con las estructruras de pensamiento del sujeto. La abstracción lógico-algorítmica es la acción de la razón humana para organizar las reglas algorítmicas guiadas por la lógica humana a fin de resolver un problema del mudo real. Adicionalmente, el diseño de algoritmos es la escritura de las estructuras lógicas o reglas algorítmicas, que guiadas por las reglas de la razón humana, son constructos válidos para ser codificados en lógicas de programación, y que representan la resolución del problema del usuario mediante la utilización integrada del hardware-software; hardware representado por el ordenador y el software o lógica de programación generada a partir de la lógica algorítmica.

- Las estructuras lógicas básicas para la construcción de algoritmos interactúan

con estructuras de datos, como es el caso de los arreglos unidimensionales, bidimensionales y $n$-dimensionales, para generar posibilidades de almacenamiento y procesamiento más complejos que en la medida de su avance modelen el comportamiento de los fenómenos del mundo real.

- La integración de las estructuras lógicas algorítmicas con las estructuras de datos complementadas con los conceptos de lógica de programación modular permiten la construcción de un sistema a partir de subsistemas; siendo estos los módulos que estando compuestos, a su vez, por estructuras lógicas algorítmicas se integran para constituir el sistema. Sistema funcional integrado que representa el resultado del esfuerzo y el tiempo utilizado en las acciones de las lógicas humana, algorítmica y de programación para llegar a hacer de una abstracción humana la realidad de un sistema al servicio y avance de la humanidad.

# Índice alfabético



Esta obra, editada en Barranquilla por
Editorial Universidad del Norte, se terminó de reimprimir
en los talleres de Nombre de la imprenta en mayo de 2016.
Se compuso en CMR12

Este compendio contiene las reglas básicas para diseñar algoritmos aplicados a cualquier área del conocimiento. Mediante la combinación de teoría y ejemplos, los lectores podrán identificar los conceptos de datos e información y construir algoritmos, partiendo de primitivas básicas hasta llegar a las más complejas. El texto, en síntesis, privilegia el análisis de problemas para que sean expresados con base en la lógica humana. Por ello, se encontrarán aquí diversos niveles, desde ejercicios iniciales hasta problemas de desafío, que con seguridad permitirán adquirir la destreza necesaria para construir algoritmos y aplicarlos de manera exitosa.







